```
    if (bus) {
        Dictionary d;
        d["prime"] = bus->get_bus_num();
        d["current_room"] = bus->get_current_room();
        return d;
    }

    return Variant();
}

void HilbertHotel::finish() {
    if (!thread) {
        return;
    }

    exit_thread = true;
    Thread::wait_to_finish(thread);

    memdelete(thread);

    if (mutex) {
        memdelete(mutex);
    }

    thread = NULL;
}

RID HilbertHotel::create_bus() {
    lock();
    InfiniteBus *ptr = memnew(InfiniteBus(PRIME[counter++]));
    RID ret = bus_owner.make_rid(ptr);
    ptr->set_self(ret);
    buses.insert(ret);
    unlock();

    return ret;
}

//
https://github.com/godotengine/godot/blob/master/core/templates
/rid.h
bool HilbertHotel::delete_bus(RID id) {
    if (bus_owner.owns(id)) {
        lock();
        InfiniteBus *b = bus_owner.get(id);
        bus_owner.free(id);
        buses.erase(id);
        memdelete(b);
        unlock();
        return true;
    }

    return false;
}

void HilbertHotel::clear() {
    for (Set<RID>::Element *e = buses.front(); e; e = e-
>next()) {
        delete_bus(e->get());
    }
}

bool HilbertHotel::empty() {
    return buses.size() <= 0;
}

void HilbertHotel::_bind_methods() {
}

HilbertHotel::HilbertHotel() {
    singleton = this;
}
```

```
/* prime_225.h */

const uint64_t PRIME[225] = {
        2,3,5,7,11,13,17,19,23,
        29,31,37,41,43,47,53,59,61,
        67,71,73,79,83,89,97,101,103,
        107,109,113,127,131,137,139,149,151,
        157,163,167,173,179,181,191,193,197,
        199,211,223,227,229,233,239,241,251,
        257,263,269,271,277,281,283,293,307,
        311,313,317,331,337,347,349,353,359,
        367,373,379,383,389,397,401,409,419,
        421,431,433,439,443,449,457,461,463,
        467,479,487,491,499,503,509,521,523,
        541,547,557,563,569,571,577,587,593,
        599,601,607,613,617,619,631,641,643,
        647,653,659,661,673,677,683,691,701,
        709,719,727,733,739,743,751,757,761,
        769,773,787,797,809,811,821,823,827,
        829,839,853,857,859,863,877,881,883,
        887,907,911,919,929,937,941,947,953,
        967,971,977,983,991,997,1009,1013,1019,
        1021,1031,1033,1039,1049,1051,1061,1063,1069,
        1087,1091,1093,1097,1103,1109,1117,1123,1129,
        1151,1153,1163,1171,1181,1187,1193,1201,1213,
        1217,1223,1229,1231,1237,1249,1259,1277,1279,
        1283,1289,1291,1297,1301,1303,1307,1319,1321,
        1327,1361,1367,1373,1381,1399,1409,1423,1427
};
```

Custom managed resource data

Godot servers implement a mediator pattern. All data types inherit `RID_Data`. `RID_Owner<MyRID_Data>` owns the object when `make_rid` is called. During debug mode only, RID_Owner maintains a list of RIDs. In practice, RIDs are similar to writing object-oriented C code.

```
class InfiniteBus : public RID_Data {
    RID self;

private:
    uint64_t prime_num;
    uint64_t num;

public:
    uint64_t next_room() {
        return prime_num * num++;
    }

    uint64_t get_bus_num() const {
        return prime_num;
    }

    uint64_t get_current_room() const {
        return prime_num * num;
    }

    _FORCE_INLINE_ void set_self(const RID &p_self) {
        self = p_self;
    }

    _FORCE_INLINE_ RID get_self() const {
        return self;
    }

    InfiniteBus(uint64_t prime) : prime_num(prime), num(1) {};
    ~InfiniteBus() {};
}
```

References

- RID
- core/templates/rid.h
  [https://github.com/godotengine/godot/blob/master/core/templates/rid.h]

Registering the class in GDScript

Servers are allocated in `register_types.cpp`. The constructor sets the static instance and `init()` creates the managed thread; `unregister_types.cpp` cleans up the server.

Since a Godot server class creates an instance and binds it to a static singleton, binding the class might not reference the correct instance. Therefore, a dummy class must be created to reference the proper Godot server.

In `register_server_types()`, `Engine::get_singleton()->add_singleton` is used to register the dummy class in GDScript.

```
/* register_types.cpp */

#include "register_types.h"

#include "core/object/class_db.h"
#include "core/config/engine.h"

#include "hilbert_hotel.h"

static HilbertHotel *hilbert_hotel = NULL;
static _HilbertHotel *_hilbert_hotel = NULL;

void register_hilbert_hotel_types() {
    hilbert_hotel = memnew(HilbertHotel);
    hilbert_hotel->init();
    _hilbert_hotel = memnew(_HilbertHotel);
    ClassDB::register_class<_HilbertHotel>();
    Engine::get_singleton()-
>add_singleton(Engine::Singleton("HilbertHotel",
_HilbertHotel::get_singleton()));
}

void unregister_hilbert_hotel_types() {
    if (hilbert_hotel) {
        hilbert_hotel->finish();
        memdelete(hilbert_hotel);
    }

    if (_hilbert_hotel) {
        memdelete(_hilbert_hotel);
    }
}
```

```
/* register_types.h */

/* Yes, the word in the middle must be the same as the module
folder name */
void register_hilbert_hotel_types();
void unregister_hilbert_hotel_types();
```

- servers/register_server_types.cpp [https://github.com/godotengine/godot/blob/master/servers/register_server_types.cpp]

Bind methods

The dummy class binds singleton methods to GDScript. In most cases, the dummy class methods wraps around.

```
Variant _HilbertHotel::get_bus_info(RID id) {
    return HilbertHotel::get_singleton()->get_bus_info(id);
}
```

Binding Signals

It is possible to emit signals to GDScript by calling the GDScript dummy object.

```
void HilbertHotel::_emit_occupy_room(uint64_t room, RID rid) {
    _HilbertHotel::get_singleton()->_occupy_room(room, rid);
}
```

```
class _HilbertHotel : public Object {
    GDCLASS(_HilbertHotel, Object);

    friend class HilbertHotel;
    static _HilbertHotel *singleton;

protected:
    static void _bind_methods();

private:
    void _occupy_room(int room_number, RID bus);

public:
    RID create_bus();
    void connect_signals();
    bool delete_bus(RID id);
    static _HilbertHotel *get_singleton();
    Variant get_bus_info(RID id);
    _HilbertHotel();
    ~_HilbertHotel();
};

#endif
```

```
_HilbertHotel *_HilbertHotel::singleton = NULL;
_HilbertHotel *_HilbertHotel::get_singleton() { return
singleton; }

RID _HilbertHotel::create_bus() {
    return HilbertHotel::get_singleton()->create_bus();
}

bool _HilbertHotel::delete_bus(RID rid) {
    return HilbertHotel::get_singleton()->delete_bus(rid);
}

void _HilbertHotel::_occupy_room(int room_number, RID bus) {
    emit_signal("occupy_room", room_number, bus);
}

Variant _HilbertHotel::get_bus_info(RID id) {
    return HilbertHotel::get_singleton()->get_bus_info(id);
}

void _HilbertHotel::_bind_methods() {
    ClassDB::bind_method(D_METHOD("get_bus_info", "r_id"),
&_HilbertHotel::get_bus_info);
    ClassDB::bind_method(D_METHOD("create_bus"),
&_HilbertHotel::create_bus);
    ClassDB::bind_method(D_METHOD("delete_bus"),
&_HilbertHotel::delete_bus);
    ADD_SIGNAL(MethodInfo("occupy_room",
PropertyInfo(Variant::INT, "room_number"),
PropertyInfo(Variant::_RID, "r_id")));
}

void _HilbertHotel::connect_signals() {
    HilbertHotel::get_singleton()->connect("occupy_room",
_HilbertHotel::get_singleton(), "_occupy_room");
}

_HilbertHotel::_HilbertHotel() {
    singleton = this;
}

_HilbertHotel::~_HilbertHotel() {
}
```

MessageQueue

In order to send commands into SceneTree, MessageQueue is a thread-safe buffer to queue set and call methods for other threads. To queue a command, obtain the target object RID and use either `push_call`, `push_set`, or `push_notification` to execute the desired behavior. The queue will be flushed whenever either `SceneTree::idle` or `SceneTree::iteration` is executed.

References:

- core/object/message_queue.cpp [https://github.com/godotengine/godot/blob/master/core/object/message_queue.cpp]

Summing it up

Here is the GDScript sample code:

```
extends Node

func _ready():
    print("Start debugging")
    HilbertHotel.occupy_room.connect(_print_occupy_room)
    var rid = HilbertHotel.create_bus()
    OS.delay_msec(2000)
    HilbertHotel.create_bus()
    OS.delay_msec(2000)
    HilbertHotel.create_bus()
    OS.delay_msec(2000)
    print(HilbertHotel.get_bus_info(rid))
    HilbertHotel.delete_bus(rid)
    print("Ready done")

func _print_occupy_room(room_number, r_id):
    print("Room number: "  + str(room_number) + ", RID: " + 
str(r_id))
    print(HilbertHotel.get_bus_info(r_id))
```

Notes

- The actual Hilbert Hotel [https://en.wikipedia.org/wiki/Hilbert%27s_paradox_of_the_Grand_Hotel] is impossible.
- Connecting signal example code is pretty hacky.

Custom resource format loaders

Introduction

ResourceFormatLoader is a factory interface for loading file assets. Resources are primary containers. When load is called on the same file path again, the previous loaded Resource will be referenced. Naturally, loaded resources must be stateless.

This guide assumes the reader knows how to create C++ modules and Godot data types. If not, refer to this guide: Custom modules in C++

References

- ResourceLoader
- core/io/resource_loader.cpp [https://github.com/godotengine/godot/blob/master/core/io/resource_loader.cpp]

What for?

- Adding new support for many file formats
- Audio formats
- Video formats
- Machine learning models

What not?

- Raster images

ImageFormatLoader should be used to load images.

References

- [core/io/image_loader.h](https://github.com/godotengine/godot/blob/master/core/io/image_loader.h) [https://github.com/godotengine/godot/blob/master/core/io/image_loader.h]

Creating a ResourceFormatLoader

Each file format consist of a data container and a `ResourceFormatLoader`.

ResourceFormatLoaders are classes which return all the necessary metadata for supporting new extensions in Godot. The class must return the format name and the extension string.

In addition, ResourceFormatLoaders must convert file paths into resources with the `load` function. To load a resource, `load` must read and handle data serialization.

```
/* resource_loader_json.h */

#ifndef RESOURCE_LOADER_JSON_H
#define RESOURCE_LOADER_JSON_H

#include "core/io/resource_loader.h"

class ResourceFormatLoaderJson : public ResourceFormatLoader {
    GDCLASS(ResourceFormatLoaderJson, ResourceFormatLoader);
public:
    virtual RES load(const String &p_path, const String
&p_original_path, Error *r_error = NULL);
    virtual void get_recognized_extensions(List<String>
*r_extensions) const;
    virtual bool handles_type(const String &p_type) const;
    virtual String get_resource_type(const String &p_path)
const;
};
#endif // RESOURCE_LOADER_JSON_H
```

```
/* resource_loader_json.cpp */

#include "resource_loader_json.h"

#include "resource_json.h"

RES ResourceFormatLoaderJson::load(const String &p_path, const
String &p_original_path, Error *r_error) {
Ref<JsonResource> json = memnew(JsonResource);
    if (r_error) {
        *r_error = OK;
    }
    Error err = json->load_file(p_path);
    return json;
}

void
ResourceFormatLoaderJson::get_recognized_extensions(List<String
> *r_extensions) const {
    if (!r_extensions->find("json")) {
        r_extensions->push_back("json");
    }
}

String ResourceFormatLoaderJson::get_resource_type(const String
&p_path) const {
    return "Resource";
}

bool ResourceFormatLoaderJson::handles_type(const String
&p_type) const {
    return ClassDB::is_parent_class(p_type, "Resource");
}
```

Creating a ResourceFormatSaver

If you'd like to be able to edit and save a resource, you can implement a `ResourceFormatSaver`:

```
/* resource_saver_json.h */

#ifndef RESOURCE_SAVER_JSON_H
#define RESOURCE_SAVER_JSON_H

#include "core/io/resource_saver.h"

class ResourceFormatSaverJson : public ResourceFormatSaver {
    GDCLASS(ResourceFormatSaverJson, ResourceFormatSaver);
public:
    virtual Error save(const String &p_path, const RES
&p_resource, uint32_t p_flags = 0);
    virtual bool recognize(const RES &p_resource) const;
    virtual void get_recognized_extensions(const RES
&p_resource, List<String> *r_extensions) const;
};
#endif // RESOURCE_SAVER_JSON_H
```

```
/* resource_saver_json.cpp */

#include "resource_saver_json.h"

#include "resource_json.h"
#include "scene/resources/resource_format_text.h"

Error ResourceFormatSaverJson::save(const String &p_path, const
RES &p_resource, uint32_t p_flags) {
    Ref<JsonResource> json = memnew(JsonResource);
    Error error = json->save_file(p_path, p_resource);
    return error;
}

bool ResourceFormatSaverJson::recognize(const RES &p_resource)
const {
    return Object::cast_to<JsonResource>(*p_resource) != NULL;
}

void ResourceFormatSaverJson::get_recognized_extensions(const
RES &p_resource, List<String> *r_extensions) const {
    if (Object::cast_to<JsonResource>(*p_resource)) {
        r_extensions->push_back("json");
    }
}
```

Creating custom data types

Godot may not have a proper substitute within its Core types or managed resources. Godot needs a new registered data type to understand additional binary formats such as machine learning models.

Here is an example of creating a custom datatype:

```
/* resource_json.h */

#ifndef RESOURCE_JSON_H
#define RESOURCE_JSON_H

#include "core/io/json.h"
#include "core/variant_parser.h"

class JsonResource : public Resource {
    GDCLASS(JsonResource, Resource);

protected:
    static void _bind_methods() {
        ClassDB::bind_method(D_METHOD("set_dict", "dict"),
&JsonResource::set_dict);
        ClassDB::bind_method(D_METHOD("get_dict"),
&JsonResource::get_dict);

        ADD_PROPERTY(PropertyInfo(Variant::DICTIONARY,
"content"), "set_dict", "get_dict");
    }

private:
    Dictionary content;
public:
    Error load_file(const String &p_path);
    Error save_file(const String &p_path, const RES
&p_resource);

    void set_dict(const Dictionary &p_dict);
    Dictionary get_dict();
};
#endif // RESOURCE_JSON_H
```

```
/* resource_json.cpp */

#include "resource_json.h"

Error JsonResource::load_file(const String &p_path) {
    Error error;
    FileAccess *file = FileAccess::open(p_path,
FileAccess::READ, &error);
    if (error != OK) {
        if (file) {
            file->close();
        }
        return error;
    }

    String json_string = String("");
    while (!file->eof_reached()) {
        json_string += file->get_line();
    }
    file->close();

    String error_string;
    int error_line;
    JSON json;
    Variant result;
    error = json.parse(json_string, result, error_string,
error_line);
    if (error != OK) {
        file->close();
        return error;
    }
    content = Dictionary(result);
    return OK;
}

Error JsonResource::save_file(const String &p_path, const RES
&p_resource) {
    Error error;
    FileAccess *file = FileAccess::open(p_path,
FileAccess::WRITE, &error);
    if (error != OK) {
        if (file) {
            file->close();
        }
        return error;
    }

    Ref<JsonResource> json_ref = p_resource.get_ref_ptr();
    JSON json;

    file->store_string(json.print(json_ref->get_dict(), "
"));
    file->close();
    return OK;
}

void JsonResource::set_dict(const Dictionary &p_dict) {
    content = p_dict;
}

Dictionary JsonResource::get_dict() {
    return content;
}
```

Considerations

Some libraries may not define certain common routines such as IO handling. Therefore, Godot call translations are required.

For example, here is the code for translating `FileAccess` calls into `std::istream`.

```
#include "core/os/file_access.h"

#include <istream>
#include <streambuf>

class GodotFileInStreamBuf : public std::streambuf {

public:
    GodotFileInStreamBuf(FileAccess *fa) {
        _file = fa;
    }
    int underflow() {
        if (_file->eof_reached()) {
            return EOF;
        } else {
            size_t pos = _file->get_position();
            uint8_t ret = _file->get_8();
            _file->seek(pos); // Required since get_8()
advances the read head.
            return ret;
        }
    }
    int uflow() {
        return _file->eof_reached() ? EOF : _file->get_8();
    }

private:
    FileAccess *_file;
};
```

References

- istream [https://cplusplus.com/reference/istream/istream/]
- streambuf [https://cplusplus.com/reference/streambuf/streambuf/?kw=streambuf]
- core/io/file_access.h [https://github.com/godotengine/godot/blob/master/core/os/file_access.h]

Registering the new file format

Godot registers `ResourcesFormatLoader` with a `ResourceLoader` handler. The handler selects the proper loader automatically when `load` is called.

```
/* register_types.h */

void register_json_types();
void unregister_json_types();
```

```
/* register_types.cpp */

#include "register_types.h"

#include "core/class_db.h"
#include "resource_loader_json.h"
#include "resource_saver_json.h"
#include "resource_json.h"

static Ref<ResourceFormatLoaderJson> json_loader;
static Ref<ResourceFormatSaverJson> json_saver;

void register_json_types() {
    ClassDB::register_class<JsonResource>();

    json_loader.instantiate();
    ResourceLoader::add_resource_format_loader(json_loader);

    json_saver.instantiate();
    ResourceSaver::add_resource_format_saver(json_saver);
}

void unregister_json_types() {
    ResourceLoader::remove_resource_format_loader(json_loader);
    json_loader.unref();

    ResourceSaver::remove_resource_format_saver(json_saver);
    json_saver.unref();
}
```

References

- core/io/resource_loader.cpp [https://github.com/godotengine/godot/blob/master/core/io/resource_loader.cpp]

Loading it on GDScript

Save a file called `demo.json` with the following contents and place it in the project's root folder:

```
{
  "savefilename": "demo.json",
  "demo": [
    "welcome",
    "to",
    "godot",
    "resource",
    "loaders"
  ]
}
```

Then attach the following script to any node:

```
extends Node

@onready var json_resource = load("res://demo.json")

func _ready():
    print(json_resource.get_dict())
```

Custom AudioStreams

Introduction

AudioStream is the base class of all audio emitting objects. AudioStreamPlayer binds onto an AudioStream to emit PCM data into an AudioServer which manages audio drivers.

All audio resources require two audio based classes: AudioStream and AudioStreamPlayback. As a data container, AudioStream contains the resource and exposes itself to GDScript. AudioStream references its own internal custom AudioStreamPlayback which translates AudioStream into PCM data.

This guide assumes the reader knows how to create C++ modules. If not, refer to this guide Custom modules in C++.

References:

- servers/audio/audio_stream.h [https://github.com/godotengine/godot/blob/master/servers/audio/audio_stream.h]
- scene/audio/audio_stream_player.cpp [https://github.com/godotengine/godot/blob/master/scene/audio/audio_stream_player.cpp]

What for?

- Binding external libraries (like Wwise, FMOD, etc).
- Adding custom audio queues
- Adding support for more audio formats

Create an AudioStream

An AudioStream consists of three components: data container, stream name, and an AudioStreamPlayback friend class generator. Audio data can be loaded in a number of ways such as with an internal counter for a tone generator, internal/external buffer, or a file reference.

Some AudioStreams need to be stateless such as objects loaded from ResourceLoader. ResourceLoader loads once and references the same object regardless how many times `load` is called on a specific resource. Therefore, playback state must be self-contained in AudioStreamPlayback.

```
/* audiostream_mytone.h */

#include "core/reference.h"
#include "core/resource.h"
#include "servers/audio/audio_stream.h"

class AudioStreamMyTone : public AudioStream {
    GDCLASS(AudioStreamMyTone, AudioStream)

private:
    friend class AudioStreamPlaybackMyTone;
    uint64_t pos;
    int mix_rate;
    bool stereo;
    int hz;

public:
    void reset();
    void set_position(uint64_t pos);
    virtual Ref<AudioStreamPlayback> instance_playback();
    virtual String get_stream_name() const;
    void gen_tone(int16_t *pcm_buf, int size);
    virtual float get_length() const { return 0; } // if
supported, otherwise return 0
    AudioStreamMyTone();
protected:
    static void _bind_methods();
};
```

```
/* audiostream_mytone.cpp */

#include "audiostream_mytone.h"

AudioStreamMyTone::AudioStreamMyTone()
        : mix_rate(44100), stereo(false), hz(639) {
}

Ref<AudioStreamPlayback> AudioStreamMyTone::instance_playback()
{
    Ref<AudioStreamPlaybackMyTone> talking_tree;
    talking_tree.instantiate();
    talking_tree->base = Ref<AudioStreamMyTone>(this);
    return talking_tree;
}

String AudioStreamMyTone::get_stream_name() const {
    return "MyTone";
}
void AudioStreamMyTone::reset() {
    set_position(0);
}
void AudioStreamMyTone::set_position(uint64_t p) {
    pos = p;
}
void AudioStreamMyTone::gen_tone(int16_t *pcm_buf, int size) {
    for (int i = 0; i < size; i++) {
        pcm_buf[i] = 32767.0 * sin(2.0 * Math_PI * double(pos +
i) / (double(mix_rate) / double(hz)));
    }
    pos += size;
}
void AudioStreamMyTone::_bind_methods() {
    ClassDB::bind_method(D_METHOD("reset"),
&AudioStreamMyTone::reset);
    ClassDB::bind_method(D_METHOD("get_stream_name"),
&AudioStreamMyTone::get_stream_name);
}
```

References:

- servers/audio/audio_stream.h [https://github.com/godotengine/godot/blob/master/servers/audio/audio_stream.h]

Create an AudioStreamPlayback

AudioStreamPlayer uses `mix` callback to obtain PCM data. The callback must match sample rate and fill the buffer.

Since AudioStreamPlayback is controlled by the audio thread, i/o and dynamic memory allocation are forbidden.

```
/*  audiostreamplayer_mytone.h */

#include "core/reference.h"
#include "core/resource.h"
#include "servers/audio/audio_stream.h"

class AudioStreamPlaybackMyTone : public AudioStreamPlayback {
    GDCLASS(AudioStreamPlaybackMyTone, AudioStreamPlayback)
    friend class AudioStreamMyTone;

private:
    enum {
        PCM_BUFFER_SIZE = 4096
    };
    enum {
        MIX_FRAC_BITS = 13,
        MIX_FRAC_LEN = (1 << MIX_FRAC_BITS),
        MIX_FRAC_MASK = MIX_FRAC_LEN - 1,
    };
    void *pcm_buffer;
    Ref<AudioStreamMyTone> base;
    bool active;

public:
    virtual void start(float p_from_pos = 0.0);
    virtual void stop();
    virtual bool is_playing() const;
    virtual int get_loop_count() const; // times it looped
    virtual float get_playback_position() const;
    virtual void seek(float p_time);
    virtual void mix(AudioFrame *p_buffer, float p_rate_scale,
int p_frames);
    virtual float get_length() const; // if supported,
otherwise return 0
    AudioStreamPlaybackMyTone();
    ~AudioStreamPlaybackMyTone();
};
```

```
/* audiostreamplayer_mytone.cpp */

#include "audiostreamplayer_mytone.h"

#include "core/math/math_funcs.h"
#include "core/print_string.h"

AudioStreamPlaybackMyTone::AudioStreamPlaybackMyTone()
        : active(false) {
    AudioServer::get_singleton()->lock();
    pcm_buffer = AudioServer::get_singleton()-
>audio_data_alloc(PCM_BUFFER_SIZE);
    zeromem(pcm_buffer, PCM_BUFFER_SIZE);
    AudioServer::get_singleton()->unlock();
}
AudioStreamPlaybackMyTone::~AudioStreamPlaybackMyTone() {
    if(pcm_buffer) {
        AudioServer::get_singleton()-
>audio_data_free(pcm_buffer);
        pcm_buffer = NULL;
    }
}
void AudioStreamPlaybackMyTone::stop() {
    active = false;
    base->reset();
}
void AudioStreamPlaybackMyTone::start(float p_from_pos) {
    seek(p_from_pos);
    active = true;
}
void AudioStreamPlaybackMyTone::seek(float p_time) {
    float max = get_length();
    if (p_time < 0) {
            p_time = 0;
    }
    base->set_position(uint64_t(p_time * base->mix_rate) <<
MIX_FRAC_BITS);
}
void AudioStreamPlaybackMyTone::mix(AudioFrame *p_buffer, float
p_rate, int p_frames) {
    ERR_FAIL_COND(!active);
    if (!active) {
            return;
    }
    zeromem(pcm_buffer, PCM_BUFFER_SIZE);
    int16_t *buf = (int16_t *)pcm_buffer;
    base->gen_tone(buf, p_frames);

    for(int i = 0; i < p_frames; i++) {
        float sample = float(buf[i]) / 32767.0;
        p_buffer[i] = AudioFrame(sample, sample);
    }
}
int AudioStreamPlaybackMyTone::get_loop_count() const {
    return 0;
}
float AudioStreamPlaybackMyTone::get_playback_position() const
{
    return 0.0;
}
float AudioStreamPlaybackMyTone::get_length() const {
    return 0.0;
}
bool AudioStreamPlaybackMyTone::is_playing() const {
    return active;
}
```

Resampling

Godot's AudioServer currently uses 44100 Hz sample rate. When other sample rates are needed such as 48000, either provide one or use AudioStreamPlaybackResampled. Godot provides cubic interpolation for audio resampling.

Instead of overloading `mix`, AudioStreamPlaybackResampled uses `_mix_internal` to query AudioFrames and `get_stream_sampling_rate` to query current mix rate.

```
#include "core/reference.h"
#include "core/resource.h"
#include "servers/audio/audio_stream.h"

class AudioStreamMyToneResampled;

class AudioStreamPlaybackResampledMyTone : public
AudioStreamPlaybackResampled {
    GDCLASS(AudioStreamPlaybackResampledMyTone,
AudioStreamPlaybackResampled)
    friend class AudioStreamMyToneResampled;

private:
    enum {
        PCM_BUFFER_SIZE = 4096
    };
    enum {
        MIX_FRAC_BITS = 13,
        MIX_FRAC_LEN = (1 << MIX_FRAC_BITS),
        MIX_FRAC_MASK = MIX_FRAC_LEN - 1,
    };
    void *pcm_buffer;
    Ref<AudioStreamMyToneResampled> base;
    bool active;

protected:
    virtual void _mix_internal(AudioFrame *p_buffer, int
p_frames);
public:
    virtual void start(float p_from_pos = 0.0);
    virtual void stop();
    virtual bool is_playing() const;
    virtual int get_loop_count() const; // times it looped
    virtual float get_playback_position() const;
    virtual void seek(float p_time);
    virtual float get_length() const; // if supported,
otherwise return 0
    virtual float get_stream_sampling_rate();
    AudioStreamPlaybackResampledMyTone();
    ~AudioStreamPlaybackResampledMyTone();
};
```

```
#include "mytone_audiostream_resampled.h"

#include "core/math/math_funcs.h"
#include "core/print_string.h"

AudioStreamPlaybackResampledMyTone::AudioStreamPlaybackResample
dMyTone()
        : active(false) {
    AudioServer::get_singleton()->lock();
    pcm_buffer = AudioServer::get_singleton()-
>audio_data_alloc(PCM_BUFFER_SIZE);
    zeromem(pcm_buffer, PCM_BUFFER_SIZE);
    AudioServer::get_singleton()->unlock();
}
AudioStreamPlaybackResampledMyTone::~AudioStreamPlaybackResampl
edMyTone() {
    if (pcm_buffer) {
        AudioServer::get_singleton()-
>audio_data_free(pcm_buffer);
        pcm_buffer = NULL;
    }
}
void AudioStreamPlaybackResampledMyTone::stop() {
    active = false;
    base->reset();
}
void AudioStreamPlaybackResampledMyTone::start(float
p_from_pos) {
    seek(p_from_pos);
    active = true;
}
void AudioStreamPlaybackResampledMyTone::seek(float p_time) {
    float max = get_length();
    if (p_time < 0) {
            p_time = 0;
    }
    base->set_position(uint64_t(p_time * base->mix_rate) <<
MIX_FRAC_BITS);
}
void
AudioStreamPlaybackResampledMyTone::_mix_internal(AudioFrame
*p_buffer, int p_frames) {
    ERR_FAIL_COND(!active);
    if (!active) {
        return;
    }
    zeromem(pcm_buffer, PCM_BUFFER_SIZE);
    int16_t *buf = (int16_t *)pcm_buffer;
    base->gen_tone(buf, p_frames);

    for(int i = 0;  i < p_frames; i++) {
        float sample = float(buf[i]) / 32767.0;
            p_buffer[i] = AudioFrame(sample, sample);
    }
}
float
AudioStreamPlaybackResampledMyTone::get_stream_sampling_rate()
{
    return float(base->mix_rate);
}
int AudioStreamPlaybackResampledMyTone::get_loop_count() const
{
    return 0;
}
float
AudioStreamPlaybackResampledMyTone::get_playback_position()
const {
    return 0.0;
}
float AudioStreamPlaybackResampledMyTone::get_length() const {
    return 0.0;
```

```
}
bool AudioStreamPlaybackResampledMyTone::is_playing() const {
    return active;
}
```

References:

- core/math/audio_frame.h [https://github.com/godotengine/godot/blob/master/core/math/audio_frame.h]
- servers/audio/audio_stream.h [https://github.com/godotengine/godot/blob/master/servers/audio/audio_stream.h]
- scene/audio/audio_stream_player.cpp [https://github.com/godotengine/godot/blob/master/scene/audio/audio_stream_player.cpp]

Custom platform ports

Similar to Custom modules in C++, Godot's multi-platform architecture is designed in a way that allows creating platform ports without modifying any existing source code.

An example of a custom platform port distributed independently from the engine is FRT [https://github.com/efornara/frt], which targets single-board computers. Note that this platform port currently targets Godot 3.x; therefore, it does not use the DisplayServer abstraction that is new in Godot 4.

Some reasons to create custom platform ports might be:

- You want to port your game to consoles, but wish to write the platform layer yourself. This is a long and arduous process, as it requires signing NDAs with console manufacturers, but it allows you to have full control over the console porting process.
- You want to port Godot to an exotic platform that isn't currently supported.

If you have questions about creating a custom platform port, feel free to ask in the `#platforms` channel of the Godot Contributors Chat [https://chat.godotengine.org/channel/platforms].

Note

Godot is a modern engine with modern requirements. Even if you only intend to run simple 2D projects on the target platform, it still requires an amount of memory that makes it unviable to run on most retro consoles. For reference, in Godot 4, an empty project with nothing visible requires about 100 MB of RAM to run on Linux (50 MB in headless mode).

If you want to run Godot on heavily memory-constrained platforms, older Godot versions have lower memory requirements. The porting process is similar, with the exception of DisplayServer not being split from the OS singleton.

Official platform ports

The official platform ports can be used as a reference when creating a custom platform port:

- Windows [https://github.com/godotengine/godot/tree/master/platform/windows]
- macOS [https://github.com/godotengine/godot/tree/master/platform/macos]
- Linux/*BSD [https://github.com/godotengine/godot/tree/master/platform/linuxbsd]
- Android [https://github.com/godotengine/godot/tree/master/platform/android]
- iOS [https://github.com/godotengine/godot/tree/master/platform/ios]
- Web [https://github.com/godotengine/godot/tree/master/platform/web]

While platform code is usually self-contained, there are exceptions to this rule. For instance, audio drivers that are shared across several platforms and rendering backends are located in the drivers/ folder [https://github.com/godotengine/godot/tree/master/drivers] of the Godot source code.

Creating a custom platform port

Creating a custom platform port is a large undertaking which requires prior knowledge of the platform's SDKs. Depending on what features you need, the amount of work needed varies:

Required features of a platform port

At the very least, a platform port must have methods from the OS singleton implemented to be buildable and usable for headless operation. A `logo.svg` (32×32) vector image must also be present within the platform folder. This logo is displayed in the Export dialog for each export preset targeting the platform in question.

See this implementation [https://github.com/godotengine/godot/blob/master/platform/linuxbsd/os_linuxbsd.cpp] for the Linux/*BSD platform as an example. See also the OS singleton header [https://github.com/godotengine/godot/blob/master/core/os/os.h] for reference.

Note

If your target platform is UNIX-like, consider inheriting from the `OS_Unix` class to get much of the work done automatically.

If the platform is not UNIX-like, you might use the *Windows port <https://github.com/godotengine/godot/blob/master/platform/windows/os_windows.cpp>* as a reference.

detect.py file

A `detect.py` file must be created within the platform's folder with all methods implemented. This file is required for SCons to detect the platform as a valid option for compiling. See the detect.py file [https://github.com/godotengine/godot/blob/master/platform/linuxbsd/detect.py] for the Linux/*BSD platform as an example.

All methods should be implemented within `detect.py` as follows:

- `is_active()`: Can be used to temporarily disable building for a platform. This should generally always return `True`.
- `get_name()`: Returns the platform's user-visible name as a string.
- `can_build()`: Return `True` if the host system is able to build for the target platform, `False` otherwise. Do not put slow checks here, as this is queried when the list of platforms is requested by the user. Use `configure()` for extensive dependency checks instead.
- `get_opts()`: Returns the list of SCons build options that can be defined by the user for this platform.
- `get_flags()`: Returns the list of overridden SCons flags for this platform.
- `configure()`: Perform build configuration, such as selecting compiler options depending on SCons options chosen.

Optional features of a platform port

In practice, headless operation doesn't suffice if you want to see anything on screen and handle input devices. You may also want audio output for most games.

*Some links on this list point to the Linux/*BSD platform implementation as a reference.*

- One or more DisplayServers [https://github.com/godotengine/godot/blob/master/platform/linuxbsd/x11/display_server_x11.cpp], with the windowing methods implemented. DisplayServer also covers features such as mouse support, touchscreen support and tablet driver (for pen input). See the DisplayServer singleton header [https://github.com/godotengine/godot/blob/master/servers/display_server.h] for reference.
    - For platforms not featuring full windowing support (or if it's not relevant for the port you are making), most windowing functions can be left mostly unimplemented. These

functions can be made to only check if the window ID is `MAIN_WINDOW_ID` and specific operations like resizing may be tied to the platform's screen resolution feature (if relevant). Any attempt to create or manipulate other window IDs can be rejected.

- *If the target platform supports the graphics APIs in question:* Rendering context for Vulkan [https://github.com/godotengine/godot/blob/master/platform/linuxbsd/x11/vulkan_context_x11.cpp], OpenGL 3.3 or OpenGL ES 3.0 [https://github.com/godotengine/godot/blob/master/platform/linuxbsd/x11/gl_manager_x11.cpp].
- Input handlers for keyboard [https://github.com/godotengine/godot/blob/master/platform/linuxbsd/x11/key_mapping_x11.cpp] and controller [https://github.com/godotengine/godot/blob/master/platform/linuxbsd/joypad_linux.cpp].
- One or more audio drivers [https://github.com/godotengine/godot/blob/master/drivers/pulseaudio/audio_driver_pulseaudio.cpp]. The audio driver can be located in the `platform/` folder (this is done for the Android and Web platforms), or in the `drivers/` folder if multiple platforms may be using this audio driver. See the AudioServer singleton header [https://github.com/godotengine/godot/blob/master/servers/audio_server.h] for reference.
- Crash handler [https://github.com/godotengine/godot/blob/master/platform/linuxbsd/crash_handler_linuxbsd.cpp], for printing crash backtraces when the game crashes. This allows for easier troubleshooting on platforms where logs aren't readily accessible.
- Text-to-speech driver [https://github.com/godotengine/godot/blob/master/platform/linuxbsd/tts_linux.cpp] (for accessibility).
- Export handler [https://github.com/godotengine/godot/tree/master/platform/linuxbsd/export] (for exporting from the editor, including One-click deploy). Not

required if you intend to export only a PCK from the editor, then run the export template binary directly by renaming it to match the PCK file. See the EditorExportPlatform header [https://github.com/godotengine/godot/blob/master/editor/export/editor_export_platform.h] for reference. `run_icon.svg` (16×16) should be present within the platform folder if One-click deploy is implemented for the target platform. This icon is displayed at the top of the editor when one-click deploy is set up for the target platform.

If the target platform doesn't support running Vulkan, OpenGL 3.3 or OpenGL ES 3.0, you have two options:

- Use a library at run-time to translate Vulkan or OpenGL calls to another graphics API. For example, MoltenVK [https://moltengl.com/moltenvk/] is used on macOS to translate Vulkan to Metal at run-time.
- Create a new renderer from scratch. This is a large undertaking, especially if you want to support both 2D and 3D rendering with advanced features.

Distributing a custom platform port

Warning

Before distributing a custom platform port, make sure you're allowed to distribute all the code that is being linked against. Console SDKs are typically under NDAs which prevent redistribution to the public.

Platform ports are designed to be as self-contained as possible. Most of the code can be kept within a single folder located in `platform/`. Like Custom modules in C++, this allows for streamlining the build process by making it possible to `git clone` a platform folder within a

Godot repository clone's `platform/` folder, then run `scons platform=<name>`. No other steps are necessary for building, unless third-party platform-specific dependencies need to be installed first.

However, when a custom rendering backend is needed, another folder must be added in `drivers/`. In this case, the platform port can be distributed as a fork of the Godot repository, or as a collection of several folders that can be added over a Godot Git repository clone.

Unit testing

Godot Engine allows to write unit tests directly in C++. The engine integrates the doctest [https://github.com/onqtam/doctest] unit testing framework which gives ability to write test suites and test cases next to production code, but since the tests in Godot go through a different `main` entry point, the tests reside in a dedicated `tests/` directory instead, which is located at the root of the engine source code.

Platform and target support

C++ unit tests can be run on Linux, macOS, and Windows operating systems.

Tests can only be run with editor `tools` enabled, which means that export templates cannot be tested currently.

Running tests

Before tests can be actually run, the engine must be compiled with the `tests` build option enabled (and any other build option you typically use), as the tests are not compiled as part of the engine by default:

```
scons tests=yes
```

Once the build is done, run the tests with a `--test` command-line option:

```
./bin/<godot_binary> --test
```

The test run can be configured with the various doctest-specific command-line options. To retrieve the full list of supported options, run the `--test` command with the `--help` option:

```
./bin/<godot_binary> --test --help
```

Any other options and arguments after the `--test` command are treated as arguments for doctest.

Note

Tests are compiled automatically if you use the `dev_mode=yes` SCons option. `dev_mode=yes` is recommended if you plan on contributing to the engine development as it will automatically treat compilation warnings as errors. The continuous integration system will fail if any compilation warnings are detected, so you should strive to fix all warnings before opening a pull request.

Filtering tests

By default, all tests are run if you don't supply any extra arguments after the `--test` command. But if you're writing new tests or would like to see the successful assertions output coming from those tests for debugging purposes, you can run the tests of interest with the various filtering options provided by doctest.

The wildcard syntax `*` is supported for matching any number of characters in test suites, test cases, and source file names:

Filter options	Shorthand	Examples
`--test-suite`	`-ts`	`-ts="*[GDScript]*"`

`--test-case`	`-tc`	`-tc="*[String]*"`
`--source-file`	`-sf`	`-sf="*test_color*"`

For instance, to run only the `String` unit tests, run:

```
./bin/<godot_binary> --test --test-case="*[String]*"
```

Successful assertions output can be enabled with the `--success` (`-s`) option, and can be combined with any combination of filtering options above, for instance:

```
./bin/<godot_binary> --test --source-file="*test_color*" --success
```

Specific tests can be skipped with corresponding `-exclude` options. As of now, some tests include random stress tests which take a while to execute. In order to skip those kind of tests, run the following command:

```
./bin/<godot_binary> --test --test-case-exclude="*[Stress]*"
```

Writing tests

Test suites represent C++ header files which must be included as part of the main test entry point in `tests/test_main.cpp`. Most test suites are located directly under `tests/` directory.

All header files are prefixed with `test_`, and this is a naming convention which the Godot build system relies on to detect tests throughout the engine.

Here's a minimal working test suite with a single test case written:

```
#ifndef TEST_STRING_H
#define TEST_STRING_H

#include "tests/test_macros.h"

namespace TestString {

TEST_CASE("[String] Hello World!") {
    String hello = "Hello World!";
    CHECK(hello == "Hello World!");
}

} // namespace TestString

#endif // TEST_STRING_H
```

Note

You can quickly generate new tests using the `create_test.py` script found in the `tests/` directory. This script automatically creates a new test file with the required boilerplate code in the appropriate location. It's also able to automatically include the new header in `tests/test_main.cpp` using invasive mode (`-i` flag). To view usage instructions, run the script with the `-h` flag.

The `tests/test_macros.h` header encapsulates everything which is needed for writing C++ unit tests in Godot. It includes doctest assertion and logging macros such as `CHECK` as seen above, and of course the definitions for writing test cases themselves.

See also

tests/test_macros.h [https://github.com/godotengine/godot/blob/master/tests/test_macros.h] source code for currently implemented macros and aliases for them.

Test cases are created using `TEST_CASE` function-like macro. Each test case must have a brief description written in parentheses, optionally including custom tags which allow to filter the tests at run-time, such as `[String]`, `[Stress]` etc.

Test cases are written in a dedicated namespace. This is not required, but allows to prevent naming collisions for when other static helper functions are written to accommodate the repeating testing procedures such as populating common test data for each test, or writing parameterized tests.

Godot supports writing tests per C++ module. For instructions on how to write module tests, refer to [Writing custom unit tests]().

Subcases

In situations where you have a common setup for several test cases with only slight variations, subcases can be very helpful. Here's an example:

```
TEST_CASE("[SceneTree][Node] Testing node operations with a very simple scene tree") {
    // ... common setup (e.g. creating a scene tree with a few nodes)
    SUBCASE("Move node to specific index") {
        // ... setup and checks for moving a node
    }
    SUBCASE("Remove node at specific index") {
        // ... setup and checks for removing a node
    }
}
```

Each `SUBCASE` causes the `TEST_CASE` to be executed from the beginning. Subcases can be nested to an arbitrary depth, but it is advised to limit nesting to no more than one level deep.

Assertions

A list of all commonly used assertions used throughout the Godot tests, sorted by severity.

Assertion	Description
`REQUIRE`	Test if condition holds true. Fails the entire test immediately if the condition does not hold true.
`REQUIRE_FALSE`	Test if condition does not hold true. Fails the entire test immediately if the condition holds true.
`CHECK`	Test if condition holds true. Marks the test run as failing, but allow to run other assertions.
`CHECK_FALSE`	Test if condition does not hold true. Marks the test run as failing, but allow to run other assertions.
`WARN`	Test if condition holds true. Does not fail the test under any circumstance, but logs a warning if something does not hold true.
`WARN_FALSE`	Test if condition does not hold true. Does not fail the test under any circumstance, but logs a warning if something holds true.

All of the above assertions have corresponding `*_MESSAGE` macros, which allow to print optional message with rationale of what should happen.

Prefer to use `CHECK` for self-explanatory assertions and `CHECK_MESSAGE` for more complex ones if you think that it deserves a better explanation.

See also

doctest: Assertion macros [https://github.com/onqtam/doctest/blob/master/doc/markdown/assertions.md].

Logging

The test output is handled by doctest itself, and does not rely on Godot printing or logging functionality at all, so it's recommended to use dedicated macros which allow to log test output in a format written by doctest.

Macro	Description
`MESSAGE`	Prints a message.
`FAIL_CHECK`	Marks the test as failing, but continue the execution. Can be wrapped in conditionals for complex checks.
`FAIL`	Fails the test immediately. Can be wrapped in conditionals for complex checks.

Different reporters can be chosen at run-time. For instance, here's how the output can be redirected to an XML file:

```
./bin/<godot_binary> --test --source-file="*test_validate*" --
```

```
success --reporters=xml --out=doctest.txt
```

See also

[doctest: Logging macros](https://github.com/onqtam/doctest/blob/master/doc/markdown/logging.md) [https://github.com/onqtam/doctest/blob/master/doc/markdown/logging.md].

Testing failure paths

Sometimes, it's not always feasible to test for an *expected* result. With the Godot development philosophy of that the engine should not crash and should gracefully recover whenever a non-fatal error occurs, it's important to check that those failure paths are indeed safe to execute without crashing the engine.

Unexpected behavior can be tested in the same way as anything else. The only problem this creates is that the error printing shall unnecessarily pollute the test output with errors coming from the engine itself (even if the end result is successful).

To alleviate this problem, use `ERR_PRINT_OFF` and `ERR_PRINT_ON` macros directly within test cases to temporarily disable the error output coming from the engine, for instance:

```
TEST_CASE("[Color] Constructor methods") {
    ERR_PRINT_OFF;
    Color html_invalid = Color::html("invalid");
    ERR_PRINT_ON; // Don't forget to re-enable!

    CHECK_MESSAGE(html_invalid.is_equal_approx(Color()),
        "Invalid HTML notation should result in a Color with
the default values.");
}
```

Special tags in test case names

These tags can be added to the test case name to modify or extend the test environment:

Tag	Description
`[SceneTree]`	Required for test cases that rely on a scene tree with MessageQueue to be available. It also enables a mock rendering server and ThemeDB.
`[Editor]`	Like `[SceneTree]`, but with additional editor-related infrastructure available, such as EditorSettings.
`[Audio]`	Initializes the AudioServer using a mock audio driver.
`[Navigation]`	Creates the default 2D/3D navigation servers and makes them available for testing.

You can use them together to combine multiple test environment extensions.

Testing signals

The following macros can be use to test signals:

Macro	Description
`SIGNAL_WATCH(object, "signal_name")`	Starts watching the specified signal on the given object.

Macro	Description
`SIGNAL_UNWATCH(object, "signal_name")`	Stops watching the specified signal on the given object.
`SIGNAL_CHECK("signal_name", Vector<Vector<Variant>>)`	Checks the arguments of all fired signals. The outer vector contains each fired signal, while the inner vector contains the list of arguments for that signal. The order of signals is significant.
`SIGNAL_CHECK_FALSE("signal_name")`	Checks if the specified signal was not fired.
`SIGNAL_DISCARD("signal_name")`	Discards all records of the specified signal.

Below is an example demonstrating the use of these macros:

```
//...
SUBCASE("[Timer] Timer process timeout signal must be emitted")
{
    SIGNAL_WATCH(test_timer, SNAME("timeout"));
    test_timer->start(0.1);

    SceneTree::get_singleton()->process(0.2);

    Array signal_args;
    signal_args.push_back(Array());

    SIGNAL_CHECK(SNAME("timeout"), signal_args);

    SIGNAL_UNWATCH(test_timer, SNAME("timeout"));
}
//...
```

Test tools

Test tools are advanced methods which allow you to run arbitrary procedures to facilitate the process of manual testing and debugging the engine internals.

These tools can be run by supplying the name of a tool after the `--test` command-line option. For instance, the GDScript module implements and registers several tools to help the debugging of the tokenizer, parser, and compiler:

```
./bin/<godot_binary> --test gdscript-tokenizer test.gd
./bin/<godot_binary> --test gdscript-parser test.gd
./bin/<godot_binary> --test gdscript-compiler test.gd
```

If any such tool is detected, then the rest of the unit tests are skipped.

Test tools can be registered anywhere throughout the engine as the registering mechanism closely resembles of what doctest provides while registering test cases using dynamic initialization technique, but usually these can be registered at corresponding `register_types.cpp` sources (per module or core).

Here's an example of how GDScript registers test tools in `modules/gdscript/register_types.cpp`:

```
#ifdef TESTS_ENABLED
void test_tokenizer() {
    TestGDScript::test(TestGDScript::TestType::TEST_TOKENIZER);
}

void test_parser() {
    TestGDScript::test(TestGDScript::TestType::TEST_PARSER);
}

void test_compiler() {
    TestGDScript::test(TestGDScript::TestType::TEST_COMPILER);
}

REGISTER_TEST_COMMAND("gdscript-tokenizer", &test_tokenizer);
REGISTER_TEST_COMMAND("gdscript-parser", &test_parser);
REGISTER_TEST_COMMAND("gdscript-compiler", &test_compiler);
#endif
```

The custom command-line parsing can be performed by a test tool itself with the help of OS get_cmdline_args method.

Integration tests for GDScript

Godot uses doctest to prevent regressions in GDScript during development. There are several types of test scripts which can be written:

- tests for expected errors;
- tests for warnings;
- tests for features.

Therefore, the process of writing integration tests for GDScript is the following:

1. Pick a type of a test script you'd like to write, and create a new GDScript file under the `modules/gdscript/tests/scripts` directory within corresponding sub-directory.

2. Write GDScript code. The test script must have a function called `test()` which takes no arguments. Such function will be called by the test runner. The test should not have any dependency unless it's part of the test too. Global classes (using `class_name`) are registered before the runner starts, so those should work if needed.

   Here's an example test script:

   ```
   func test():
       if true # Missing colon here.
           print("true")
   ```

3. Change directory to the Godot source repository root.

   ```
   cd godot
   ```

4. Generate `*.out` files to update the expected results from the output:

   ```
   bin/<godot_binary> --gdscript-generate-tests
   modules/gdscript/tests/scripts
   ```

You may add the `--print-filenames` option to see filenames as their test outputs are generated. If you are working on a new feature that is causing hard crashes, you can use this option to quickly find which test file causes the crash and debug from there.

5. Run GDScript tests with:

   ```
   ./bin/<godot_binary> --test --test-suite="*GDScript*"
   ```

This also accepts the `--print-filenames` option (see above).

If no errors are printed and everything goes well, you're done!

Warning

Make sure the output does have the expected values before submitting a pull request. If `--gdscript-generate-tests` produces `*.out` files which are unrelated to newly added tests, you should revert those files back and only commit `*.out` files for new tests.

Note

The GDScript test runner is meant for testing the GDScript implementation, not for testing user scripts nor testing the engine using scripts. We recommend writing new tests for already resolved issues related to GDScript at GitHub [https://github.com/godotengine/godot/issues?q=is%3Aissue+label%3Atopic%3Agdscript+is%3Aclosed], or writing tests for currently working features.

Note

If your test case requires that there is no `test()` function present inside the script file, you can disable the runtime section of the test by naming the script file so that it matches the pattern `*.notest.gd`. For example, "test_empty_file.notest.gd".

Godot file formats

- GDScript grammar
- TSCN file format

GDScript grammar

This is the formal grammar of GDScript written in EBNF [https://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_form], for reference purposes.

Note

This grammar is descriptive only, derived from the reference documentation and current implementation. The GDScript parser is **not** generated from a grammar definition. Inconsistencies here likely mean an error in this grammar, not a bug in GDScript.

```
(* GDScript EBNF grammar.
   Uppercase words are terminals generated by the tokenizer.
   INDENT/DEDENT are not generated by the tokenizer yet, but
they are added
   here for reading convenience.
   Naturally, this only cover syntax. Semantics can't be
inferred from this
   description.
*)

program = [ inheritance NEWLINE ] [ className ] { topLevelDecl
} ;

inheritance = "extends" ( IDENTIFIER | STRING ) { "."
IDENTIFIER } ;
className = "class_name" IDENTIFIER [ "," STRING ] NEWLINE ;

topLevelDecl
    = classVarDecl
    | constDecl
    | signalDecl
    | enumDecl
    | methodDecl
    | constructorDecl
    | innerClass
    | "tool"
    ;

classVarDecl = [ "onready" ] [ export ] "var" IDENTIFIER [ ":"
typeHint ]
    [ "=" expression ] [ setget ] NEWLINE ;
setget = "setget" [ IDENTIFIER ] [ "," IDENTIFIER] ;
export = "export" [ "(" [ BUILTINTYPE | IDENTIFIER { ","
literal } ] ")" ] ;
typeHint = BUILTINTYPE | IDENTIFIER ;

constDecl = "const" IDENTIFIER [ ":" typeHint ] "=" expression
NEWLINE ;

signalDecl = "signal" IDENTIFIER [ signalParList ] NEWLINE ;
signalParList = "(" [ IDENTIFIER { "," IDENTIFIER } ] ")" ;

enumDecl = "enum" [ IDENTIFIER ] "{" [ IDENTIFIER [ "=" INTEGER
]
    { "," IDENTIFIER [ "=" INTEGER ] } [ "," ] ] "}" NEWLINE ;

methodDecl = [ rpc ] [ "static" ] "func" IDENTIFIER "(" [
parList ] ")"
    [ "->" typeHint] ":" stmtOrSuite ;
parList = parameter { "," parameter } ;
parameter = [ "var" ] IDENTIFIER [ ":" typeHint ] [ "="
expression ] ;
rpc = "remote" | "master" | "puppet"
    | "remotesync" | "mastersync"  | "puppetsync";

constructorDecl = "func" IDENTIFIER "(" [ parList ] ")"
    [ "." "(" [ argList ] ")" ] ":" stmtOrSuite ;
argList = expression { "," expression } ;

innerClass = "class" IDENTIFIER [ inheritance ] ":" NEWLINE
    INDENT [ inheritance NEWLINE ] topLevelDecl { topLevelDecl
} DEDENT ;

stmtOrSuite = stmt | NEWLINE INDENT suite DEDENT ;
suite = stmt { stmt };

stmt
```

```
    = varDeclStmt
    | ifStmt
    | forStmt
    | whileStmt
    | matchStmt
    | flowStmt
    | assignmentStmt
    | exprStmt
    | assertStmt
    | yieldStmt
    | preloadStmt
    | "breakpoint" stmtEnd
    | "pass" stmtEnd
    ;
stmtEnd = NEWLINE | ";" ;

ifStmt = "if" expression ":" stmtOrSuite { "elif" expression
":" stmtOrSuite }
    [ "else" ":" stmtOrSuite ] ;
whileStmt = "while" expression ":" stmtOrSuite;
forStmt = "for" IDENTIFIER "in" expression ":" stmtOrSuite ;

matchStmt = "match" expression ":" NEWLINE INDENT matchBlock
DEDENT;
matchBlock = patternList ":" stmtOrSuite { patternList ":"
stmtOrSuite };
patternList = pattern { "," pattern } ;
(* Note: you can't have a binding in a pattern list, but to not
complicate the
grammar more it won't be restricted syntactically *)
pattern = literal | BUILTINTYPE | CONSTANT | "_" |
bindingPattern
    | arrayPattern | dictPattern ;
bindingPattern = "var" IDENTIFIER ;
arrayPattern = "[" [ pattern { "," pattern } [ ".." ] ] "]" ;
dictPattern = "{" [ keyValuePattern ] { "," keyValuePattern } [
".." ] "}" ;
keyValuePattern = STRING [ ":" pattern ] ;

flowStmt
    = "continue" stmtEnd
    | "break" stmtEnd
    | "return" [ expression ] stmtEnd
    ;

assignmentStmt = subscription ( "=" | "+=" | "-=" | "*=" | "/="
| "%=" | "&=" | "|=" | "^=" ) expression stmtEnd;
varDeclStmt = "var" IDENTIFIER [ "=" expression ] stmtEnd;

assertStmt = "assert" "(" expression [ "," STRING ] ")" stmtEnd
;
yieldStmt = "yield" "(" [ expression "," expression ] ")" ;
preloadStmt = "preload" "(" CONSTANT ")" ;

(* This expression grammar encodes precedence. Items later in
the list have
higher precedence than the ones before. *)
exprStmt = expression stmtEnd ;
expression = cast [ "[" expression "]" ] ;
cast = ternaryExpr [ "as" typeHint ];
ternaryExpr = logicOr [ "if" logicOr "else" logicOr ] ;
logicOr = logicAnd { ( "or" | "||" ) logicAnd } ;
logicAnd = logicNot { ( "and" | "&&" ) logicNot };
logicNot = ( "!" | "not" ) logicNot | in;
in = comparison { "in" comparison };
comparison = bitOr { ( "<" | ">" | "<=" | ">=" | "==" | "!=" )
bitOr } ;
bitOr = bitXor { "|" bitXor } ;
bitXor = bitAnd { "^" bitAnd } ;
bitAnd = bitShift { "&" bitShift } ;
bitShift = minus { ( "<<" | ">>" ) minus } ;
minus = plus { "-" plus } ;
plus = factor { "+" factor } ;
factor = sign { ( "*" | "/" | "%" ) sign } ;
sign = ( "-" | "+" ) sign | bitNot ;
bitNot = "~" bitNot | is ;
is = call [ "is" ( IDENTIFIER | BUILTINTYPE ) ] ;
call
    = (attribute [ "(" [ argList ] ")" ])
    | "." IDENTIFIER "(" [ argList ] ")"
    | "$" ( STRING | IDENTIFIER { '/' IDENTIFIER } );
attribute = subscription { "." IDENTIFIER } ;
subscription = primary [ "[" expression "]" ] ;
primary = "true" | "false" | "null" | "self" | literal |
arrayDecl
    | dictDecl | "(" expression ")" ;
```

```
literal = STRING | NUMBER | IDENTIFIER | BUILTINTYPE
    | "PI" | "TAU" | "NAN" | "INF" ;
arrayDecl = "[" [ expression { "," expression } "," ] "]" ;
dictDecl = "{" [ keyValue { "," keyValue } "," ] "}" ;
keyValue
    = expression ":" expression
    | IDENTIFIER "=" expression
    ;
```

TSCN file format

The TSCN (text scene) file format represents a single scene tree inside Godot. Unlike binary SCN files, TSCN files have the advantage of being mostly human-readable and easy for version control systems to manage.

The ESCN (exported scene) file format is identical to the TSCN file format, but is used to indicate to Godot that the file has been exported from another program and should not be edited by the user from within Godot. Unlike SCN and TSCN files, during import, ESCN files are compiled to binary SCN files stored inside the `.godot/imported/` folder. This reduces the data size and speeds up loading, as binary formats are faster to load compared to text-based formats.

To make files more compact, properties equal to the default value are not stored in scene/resource files. It is possible to write them manually, but they will be discarded when saving the file.

For those looking for a complete description, the parsing is handled in the file resource_format_text.cpp [https://github.com/godotengine/godot/blob/master/scene/resources/resource_format_text.cpp] in the `ResourceFormatLoaderText` class.

Note

The scene and resource file formats have changed significantly in Godot 4, with the introduction of string-based UIDs to replace incremental integer IDs.

Mesh, skeleton and animation data is also stored differently compared to Godot 3. You can read about some of the changes in this article: Animation data rework for 4.0 [https://godotengine.org/article/animation-data-redesign-40/]

Scenes and resources saved with Godot 4.x contain `format=3` in their header, whereas Godot 3.x uses `format=2` instead.

File structure

There are five main sections inside the TSCN file:

0. File descriptor
1. External resources
2. Internal resources
3. Nodes
4. Connections

The file descriptor looks like `[gd_scene load_steps=4 format=3 uid="uid://cecaux1sm7mo0"]` and should be the first entry in the file. The `load_steps` parameter is equal to the total amount of resources (internal and external) plus one (for the file itself). If the file has no resources, `load_steps` is omitted. The engine will still load the file correctly if `load_steps` is incorrect, but this will affect loading bars and any other piece of code relying on that value.

`uid` is a unique string-based identifier representing the scene. This is used by the engine to track files that are moved around, even while the editor is closed. Scripts can also load UID-based resources using the `uid://` path prefix to avoid relying on filesystem paths. This makes it possible to move around a file in the project, but still be able to load it in scripts without having to modify the script. Godot does not use external files to keep track of IDs, which means no central metadata storage location is required within the project. See this pull request [https://github.com/godotengine/godot/pull/50786] for detailed information.

These sections should appear in order, but it can be hard to distinguish them. The only difference between them is the first element in the heading for all of the items in the section. For example, the heading of all external resources should start with `[ext_resource ...]`.

A TSCN file may contain single-line comments starting with a semicolon (`;`). However, comments will be discarded when saving the file using the Godot editor. Whitespace within a TSCN file is not significant (except within strings), but extraneous whitespace will be discarded when saving the file.

Entries inside the file

A heading looks like `[<resource_type> key1=value1 key2=value2 key3=value3 ...]` where resource_type is one of:

- `ext_resource`
- `sub_resource`
- `node`
- `connection`

Below every heading comes zero or more `key = value` pairs. The values can be complex datatypes such as Arrays, Transforms, Colors, and so on. For example, a Node3D looks like:

```
[node name="Cube" type="Node3D"]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 2, 3)
```

The scene tree

The scene tree is made up of… nodes! The heading of each node consists of its name, parent and (most of the time) a type. For example: `[node name="PlayerCamera" type="Camera" parent="Player/Head"]`

Other valid keywords include:

- `instance`
- `instance_placeholder`
- `owner`
- `index` (sets the order of appearance in the tree; if absent, inherited nodes will take precedence over plain ones)
- `groups`

The first node in the file, which is also the scene root, must **not** have a `parent="Path/To/Node"` entry in its heading. All scene files should have exactly *one* scene root. If it doesn't, Godot will fail to import the file. The parent path of other nodes should be absolute, but shouldn't contain the scene root's name. If the node is a direct child of the scene root, the path should be `"."`. Here is an example scene tree (but without any node content):

```
[node name="Player" type="Node3D"]                    ; The
scene root
[node name="Arm" type="Node3D" parent="."]            ;
Parented to the scene root
[node name="Hand" type="Node3D" parent="Arm"]         ; Child
of "Arm"
[node name="Finger" type="Node3D" parent="Arm/Hand"]  ; Child
of "Hand"
```

Tip

To make the file structure easier to grasp, you can save a file with any given node or resource and then inspect it yourself in an external editor. You can also make incremental changes in the Godot editor, and keep an external text editor open on the `.tscn` or `.tres` file with auto-reload enabled to see what changes.

Here is an example of a scene containing a RigidBody3D-based ball with collision, visuals (mesh + light) and a camera parented to the

RigidBody3D:

```
[gd_scene load_steps=4 format=3 uid="uid://cecaux1sm7mo0"]

[sub_resource type="SphereShape3D" id="SphereShape3D_tj6p1"]

[sub_resource type="SphereMesh" id="SphereMesh_4w3ye"]

[sub_resource type="StandardMaterial3D"
id="StandardMaterial3D_k54se"]
albedo_color = Color(1, 0.639216, 0.309804, 1)

[node name="Ball" type="RigidBody3D"]

[node name="CollisionShape3D" type="CollisionShape3D"
parent="."]
shape = SubResource("SphereShape3D_tj6p1")

[node name="MeshInstance3D" type="MeshInstance3D" parent="."]
mesh = SubResource("SphereMesh_4w3ye")
surface_material_override/0 =
SubResource("StandardMaterial3D_k54se")

[node name="OmniLight3D" type="OmniLight3D" parent="."]
light_color = Color(1, 0.698039, 0.321569, 1)
omni_range = 10.0

[node name="Camera3D" type="Camera3D" parent="."]
transform = Transform3D(1, 0, 0, 0, 0.939693, 0.34202, 0,
-0.34202, 0.939693, 0, 1, 3)
```

NodePath

A tree structure is not enough to represent the whole scene. Godot uses a `NodePath(Path/To/Node)` structure to refer to another node or attribute of the node anywhere in the scene tree. Paths are relative to the current node, with `NodePath(".")` pointing to the current node and `NodePath("")` pointing to no node at all.

For instance, MeshInstance3D uses `NodePath()` to point to its skeleton. Likewise, Animation tracks use `NodePath()` to point to node properties to animate.

NodePath can also point to a property using a `:property_name` suffix, and even point to a specific component for vector, transform and color types. This is used by Animation resources to point to specific properties to animate. For example, `NodePath("MeshInstance3D:scale.x")` points to the `x` component of the `scale` Vector3 property in MeshInstance3D.

For example, the `skeleton` property in the MeshInstance3D node called `mesh` points to its parent, `Armature01`:

```
[node name="mesh" type="MeshInstance3D" parent="Armature01"]
skeleton = NodePath("..")
```

Skeleton3D

The Skeleton3D node inherits the Node3D node, but may also have a list of bones described in key-value pairs in the format `bones/<id>/<attribute> = value`. The bone attributes consist of:

- `position`: Vector3
- `rotation`: Quaternion
- `scale`: Vector3

These attributes are all optional. For instance, a bone may only define `position` or `rotation` without defining the other properties.

Here's an example of a skeleton node with two bones:

```
[node name="Skeleton3D" type="Skeleton3D"
parent="PlayerModel/Robot_Skeleton" index="0"]
bones/1/position = Vector3(0.114471, 2.19771, -0.197845)
bones/1/rotation = Quaternion(0.191422, -0.0471201,
-0.00831942, 0.980341)
```

```
bones/2/position = Vector3(-2.59096e-05, 0.236002, 0.000347473)
bones/2/rotation = Quaternion(-0.0580488, 0.0310587,
-0.0085914, 0.997794)
bones/2/scale = Vector3(0.9276, 0.9276, 0.9276)
```

BoneAttachment3D

The BoneAttachment3D node is an intermediate node to describe some node being parented to a single bone in a Skeleton node. The BoneAttachment has a `bone_name = "name of bone"` property, as well as a property for the matching bone index.

An example of a Marker3D node parented to a bone in Skeleton:

```
[node name="GunBone" type="BoneAttachment3D"
parent="PlayerModel/Robot_Skeleton/Skeleton3D" index="5"]
transform = Transform3D(0.333531, 0.128981, -0.933896,
0.567174, 0.763886, 0.308015, 0.753209, -0.632331, 0.181604,
-0.323915, 1.07098, 0.0497144)
bone_name = "hand.R"
bone_idx = 55

[node name="ShootFrom" type="Marker3D"
parent="PlayerModel/Robot_Skeleton/Skeleton3D/GunBone"]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0.4, 0)
```

AnimationPlayer

The AnimationPlayer node works with one or more animation libraries stored in AnimationLibrary resources. An animation library is a collection of individual Animation resources, whose structure is documented here.

This split between animations themselves and animation libraries was done in Godot 4, so that animations can be imported separately from 3D meshes, which is a common workflow in 3D animation software. See the original pull request [https://github.com/godotengine/godot/pull/59980] for details.

If the library name is empty, then it acts acts the unique source of animations for this AnimationPlayer. This allows using `<animation_name>` directly to play animations from script. If you name the library, then you must play it as `<library_name>/<animation_name>`. This ensures backwards compatibility and keeps the existing workflow if you don't want to use multiple animation libraries.

Resources

Resources are components that make up the nodes. For example, a MeshInstance3D node will have an accompanying ArrayMesh resource. The ArrayMesh resource may be either internal or external to the TSCN file.

References to the resources are handled by unique string-based IDs in the resource's heading. This is different from the `uid` property, which each external resource also has (but subresources don't).

External resources and internal resources are referred to with `ExtResource("id")` and `SubResource("id")`, respectively. Because there have different methods to refer to internal and external resources, you can have the same ID for both an internal and external resource.

For example, to refer to the resource `[ext_resource type="Material" uid="uid://c4cp0al3ljsjv" path="res://material.tres" id="1_7bt6s"]`, you would use `ExtResource("1_7bt6s")`.

External resources

External resources are links to resources not contained within the TSCN file itself. An external resource consists of a path, a type, a UID (used to map its filesystem location to a unique identifier) and an ID (used to refer to the resource in the scene file).

Godot always generates absolute paths relative to the resource directory and thus prefixed with `res://`, but paths relative to the TSCN file's location are also valid.

Some example external resources are:

```
[ext_resource type="Texture2D" uid="uid://ccbm14ebjmpy1" path="res://gradient.tres" id="2_eorut"]
[ext_resource type="Material" uid="uid://c4cp0al3ljsjv" path="material.tres" id="1_7bt6s"]
```

Like TSCN files, a TRES file may contain single-line comments starting with a semicolon (`;`). However, comments will be discarded when saving the resource using the Godot editor. Whitespace within a TRES file is not significant (except within strings), but extraneous whitespace will be discarded when saving the file.

Internal resources

A TSCN file can contain meshes, materials and other data. These are contained in the *internal resources* section of the file. The heading for an internal resource looks similar to those of external resources, except that it doesn't have a path. Internal resources also have `key=value` pairs under each heading. For example, a capsule collision shape looks like:

```
[sub_resource type="CapsuleShape3D" id="CapsuleShape3D_fdxgg"]
radius = 1.0
height = 3.0
```

Some internal resources contain links to other internal resources (such as a mesh having a material). In this case, the referring resource must appear *before* the reference to it. This means that order matters in the file's internal resources section.

ArrayMesh

An ArrayMesh consists of several surfaces contained in the `_surfaces` array (notice the leading underscore). Each surface's data is stored in a dictionary with the following keys:

- `aabb`: The computed axis-aligned bounding box for visibility.
- `attribute_data`: Vertex attribute data, such as normals, tangents, vertex colors, UV1, UV2 and custom vertex data.
- `bone_aabbs`: The axis-aligned bounding box of each bone for visibility.
- `format`: The surface's buffer format.
- `index_count`: The number of indices in the surface. This must match `index_data`'s size.
- `index_data`: The index data, which determines which vertices from `vertex_data` are drawn.
- `lods`: Level of detail variations, stored as an array. Each LOD level represents two values in the array. The first value is the percentage of screen space the LOD level is most suited for (edge length); the second value is the list of indices that should be drawn for the given LOD level.
- `material`: The material used when drawing the surface.
- `name`: The surface's name. This can be used in scripts and is imported from 3D DCCs.
- `primitive`: The surface's primitive type, matching the `Mesh.PrimitiveType` Godot enum. `0` = points, `1` = lines, `2` = line strip, `3` = triangles (most common), `4` = triangle strip.
- `skin_data`: Bone weight data.
- `vertex_count`: Number of vertices in the surface. This must match `vertex_data`'s size.
- `vertex_data`: The vertex position data.

Here's an example of an ArrayMesh saved to its own `.tres` file. Some fields were shortened with `...` for brevity:

```
[gd_resource type="ArrayMesh" load_steps=2 format=3
uid="uid://dww8o7hsqrhx5"]

[ext_resource type="Material"
path="res://player/model/playerobot.tres" id="1_r3bjq"]

[resource]
resource_name = "player_Sphere_016"
_surfaces = [{
"aabb": AABB(-0.207928, 1.21409, -0.14545, 0.415856, 0.226569,
0.223374),
"attribute_data": PackedByteArray(63, 121, ..., 117, 63),
"bone_aabbs": [AABB(0, 0, 0, -1, -1, -1), ..., AABB(-0.207928,
1.21409, -0.14545, 0.134291, 0.226569, 0.223374)],
"format": 7191,
"index_count": 1224,
"index_data": PackedByteArray(30, 0, ..., 150, 4),
"lods": [0.0382013, PackedByteArray(33, 1, ..., 150, 4)],
"material": ExtResource("1_r3bjq"),
"name": "playerobot",
"primitive": 3,
"skin_data": PackedByteArray(15, 0, ..., 0, 0),
"vertex_count": 1250,
"vertex_data": PackedByteArray(196, 169, ..., 11, 38)
}]
blend_shape_mode = 0
```

Animation

Each animation has the following properties:

- `length`: The animation's length in seconds. Note that keyframes may be placed outside the `[0; length]` interval, but they may have no effect depending on the interpolation mode chosen.
- `loop_mode`: `0` = no looping, `1` = wrap-around looping, `2` = clamped looping.

- `step`: The step size to use when editing this animation in the editor. This is only used in the editor; it doesn't affect animation playback in any way.

Each track is described by a list of key-value pairs in the format `tracks/<id>/<attribute>`. Each track includes:

- `type`: The track's type. This defines what kind of properties may be animated by this track, and how it'll be exposed to the user in the editor. Valid types are `value` (generic property track), `position_3d`, `rotation_3d`, `scale_3d`, `blend_shape` (optimized 3D animation tracks), `method` (method call tracks), `bezier` (Bezier curve tracks), `audio` (audio playback tracks), `animation` (tracks that play other animations).
- `imported`: `true` if the track was created from an imported 3D scene, `false` if it was manually created by the user in the Godot editor or using a script.
- `enabled`: `true` if the track is effective, `false` if it was disabled in the editor.
- `path`: Path to the node property that will be affected by the track. The property is written after the node path with a `:` separator.
- `interp`: The interpolation mode to use. `0` = nearest, `1` = linear, `2` = cubic, `3` = linear angle, `4` = cubic angle.
- `loop_wrap`: `true` if the track is designed to wrap around when the animation is looping, `false` if the track clamps to the first/last keyframes.
- `keys`: The animation track's values. This attribute's structure depends on the `type`.

Here is a scene containing an AnimationPlayer that scales down a cube over time using a generic property track. The AnimationLibrary workflow was not used, so the animation library has an empty name (but the animation is still given a `scale_down` name). Note that the `RESET` track was not created in this AnimationPlayer for brevity:

```
[gd_scene load_steps=4 format=3 uid="uid://cdyt3nktp6y6"]

[sub_resource type="Animation" id="Animation_r2qdp"]
resource_name = "scale_down"
length = 1.5
loop_mode = 2
step = 0.05
tracks/0/type = "value"
tracks/0/imported = false
tracks/0/enabled = true
tracks/0/path = NodePath("Box:scale")
tracks/0/interp = 1
tracks/0/loop_wrap = true
tracks/0/keys = {
"times": PackedFloat32Array(0, 1),
"transitions": PackedFloat32Array(1, 1),
"update": 0,
"values": [Vector3(1, 1, 1), Vector3(0, 0, 0)]
}

[sub_resource type="AnimationLibrary"
id="AnimationLibrary_4qx36"]
_data = {
"scale_down": SubResource("Animation_r2qdp")
}

[sub_resource type="BoxMesh" id="BoxMesh_u688r"]

[node name="Node3D" type="Node3D"]

[node name="AnimationPlayer" type="AnimationPlayer" parent="."]
autoplay = "scale_down"
libraries = {
"": SubResource("AnimationLibrary_4qx36")
}

[node name="Box" type="MeshInstance3D" parent="."]
mesh = SubResource("BoxMesh_u688r")
```

For generic property `value` tracks, `keys` is a dictionary containing 3 arrays with positions in `times` (PackedFloat32Array), easing values in `transitions` (PackedFloat32Array) and values in `values` (Array).

There is an additional `update` property, which is an integer with the values `0` = continuous, `1` = discrete, `2` = capture.

Here is a second Animation resource that makes use of the 3D Position and 3D Rotation tracks. These tracks (in addition to the 3D Scale track) replace Transform tracks from Godot 3. They are optimized for fast playback and can optionally be compressed.

The downside of these optimized track types is that they can't use custom easing values. Instead, all keyframes use linear interpolation. That said, you can still opt for using nearest or cubic interpolation for all keyframes in a given track by changing the track's interpolation mode.

```
[sub_resource type="Animation" id="Animation_r2qdp"]
resource_name = "move_and_rotate"
length = 1.5
loop_mode = 2
step = 0.05
tracks/0/type = "position_3d"
tracks/0/imported = false
tracks/0/enabled = true
tracks/0/path = NodePath("Box")
tracks/0/interp = 1
tracks/0/loop_wrap = true
tracks/0/keys = PackedFloat32Array(0, 1, 0, 0, 0, 1.5, 1, 1.5,
1, 0)
tracks/1/type = "rotation_3d"
tracks/1/imported = false
tracks/1/enabled = true
tracks/1/path = NodePath("Box")
tracks/1/interp = 1
tracks/1/loop_wrap = true
tracks/1/keys = PackedFloat32Array(0, 1, 0.211, -0.047, 0.211,
0.953, 1.5, 1, 0.005, 0.976, -0.216, 0.022)
```

For 3D position, rotation and scale tracks, `keys` is a PackedFloat32Array with all values stored in a sequence.

In the visual guide below, `T` is the keyframe's time in seconds since the start of the animation, `E` is the keyframe's transition (currently always `1`). For 3D position and scale tracks, `X`, `Y`, `Z` are the Vector3's coordinates. For 3D rotation tracks, `X`, `Y`, `Z` and `W` are the Quaternion's coordinates.

```
# For 3D position and scale, which use Vector3:
tracks/<id>/keys = PackedFloat32Array(T, E,   X, Y, Z,      T,
E,   X, Y, Z, ...)

# For 3D rotation, which use Quaternion:
tracks/<id>/keys = PackedFloat32Array(T, E,   X, Y, Z, W,
T, E,   X, Y, Z, W, ...)
```

Editor development

- Introduction to editor development
- Editor icons
- Editor style guide

Introduction to editor development

On this page, you will learn:

- The **design decisions** behind the Godot editor.
- How to work efficiently on the Godot editor's C++ code.

This guide is aimed at current or future engine contributors. To create editor plugins in GDScript, see Making plugins instead.

See also

If you are new to Godot, we recommended you to read Godot's design philosophy before continuing. Since the Godot editor is a Godot project written in C++, much of the engine's philosophy applies to the editor.

Technical choices

The Godot editor is drawn using Godot's renderer and UI system. It does *not* rely on a toolkit such as GTK or Qt. This is similar in spirit to software like Blender. While using toolkits makes it easier to achieve a "native" appearance, they are also quite heavy and their licensing is not compatible with Godot's.

The editor is fully written in C++. It can't contain any GDScript or C# code.

Directory structure

The editor's code is fully self-contained in the editor/ [https://github.com/godotengine/godot/tree/master/editor] folder of the Godot source repository.

Some editor functionality is also implemented via modules. Some of these are only enabled in editor builds to decrease the binary size of export templates. See the modules/ [https://github.com/godotengine/godot/tree/master/modules] folder in the Godot source repository.

Some important files in the editor are:

- editor/editor_node.cpp [https://github.com/godotengine/godot/blob/master/editor/editor_node.cpp] : Main editor initialization file. Effectively the "main scene" of the editor.
- editor/project_manager.cpp [https://github.com/godotengine/godot/blob/master/editor/project_manager.cpp]: Main Project Manager initialization file. Effectively the "main scene" of the Project Manager.
- editor/plugins/canvas_item_editor_plugin.cpp [https://github.com/godotengine/godot/blob/master/editor/plugins/canvas_item_editor_plugin.cpp]: The 2D editor viewport and related functionality (toolbar at the top, editing modes, overlaid helpers/panels, …).
- editor/plugins/node_3d_editor_plugin.cpp [https://github.com/godotengine/godot/blob/master/editor/plugins/node_3d_editor_plugin.cpp]: The 3D editor viewport and related functionality (toolbar at the top, editing modes, overlaid panels, …).
- editor/plugins/node_3d_editor_gizmos.cpp [https://github.com/godotengine/godot/blob/master/editor/plugins/node_3d_editor_gizmos.cpp]: Where the 3D editor gizmos are defined and drawn. This file doesn't have a 2D counterpart as 2D gizmos are drawn by the nodes themselves.

Editor dependencies in `scene/` files

When working on an editor feature, you may have to modify files in Godot's GUI nodes, which you can find in the `scene/` folder.

One rule to keep in mind is that you must **not** introduce new dependencies to `editor/` includes in other folders such as `scene/`. This applies even if you use `#ifdef TOOLS_ENABLED`.

To make the codebase easier to follow and more self-contained, the allowed dependency order is:

- `editor/` -> `scene/` -> `servers/` -> `core/`

This means that files in `editor/` can depend on includes from `scene/`, `servers/`, and `core/`. But, for example, while `scene/` can depend on includes from `servers/` and `core/`, it cannot depend on includes from `editor/`.

Currently, there are some dependencies to `editor/` includes in `scene/` files, but they are in the process of being removed [https://github.com/godotengine/godot/issues/53295].

Development tips

To iterate quickly on the editor, we recommend to set up a test project and open it from the command line after compiling the editor. This way, you don't have to go through the Project Manager every time you start Godot.

Editor icons

When a new class is created and exposed to scripting, the editor's interface will display it with a default icon representing the base class it inherits from. In most cases, it's still recommended to create icons for new classes to improve the user experience.

Creating icons

To create new icons, you first need a vector graphics editor installed. For instance, you can use the open source Inkscape [https://inkscape.org/] editor.

Clone the `godot` repository containing all the editor icons:

```
git clone https://github.com/godotengine/godot.git
```

The icons must be created in a vector graphics editor in SVG format. There are two main requirements to follow:

- Icons must be 16×16. In Inkscape, you can configure the document size in **File > Document Properties**.
- Lines should be snapped to pixels whenever possible to remain crisp at lower DPI. You can create a 16×16 grid in Inkscape to make this easier.

Once you're satisfied with the icon's design, save the icon in the cloned repository's `editor/icons` folder. The icon name should match the intended name in a case-sensitive manner. For example, to create an icon for CPUParticles2D, name the file `CPUParticles2D.svg`.

Color conversion for light editor themes

If the user has configured their editor to use a light theme, Godot will convert the icon's colors based on a set of predefined color mappings [https://github.com/godotengine/godot/blob/4.0.2-stable/editor/editor_themes.cpp#L60-L160]. This is to ensure the icon always displays with a sufficient contrast rate. Try to restrict your icon's color palette to colors found in the list above. Otherwise, your icon may become difficult to read on a light background.

Icon optimization

Because the editor renders SVGs once at load time, they need to be small in size so they can be efficiently parsed. Editor icons must be first optimized before being added to the engine, to do so:

1. Install svgcleaner [https://github.com/RazrFalcon/svgcleaner] by downloading a binary from its Releases tab [https://github.com/RazrFalcon/svgcleaner/releases/latest] and placing it into a location in your `PATH` environment variable.

2. Run the command below, replacing `svg_source.svg` with the path to your SVG file (which can be a relative or absolute path):

   ```
   svgcleaner --multipass svg_source.svg svg_optimized.svg
   ```

The `--multipass` switch improves compression, so make sure to include it. The optimized icon will be saved to `svg_optimized.svg`. You can also change the destination parameter to any relative or absolute path you'd like.

Note

While this optimization step won't impact the icon's quality noticeably, it will still remove editor-only information such as guides. Therefore, it's recommended to keep the source SVG around if you need to make further changes.

Integrating and sharing the icons

If you're contributing to the engine itself, you should make a pull request to add optimized icons to `editor/icons` in the main repository. Recompile the engine to make it pick up new icons for classes.

It's also possible to create custom icons within a module. If you're creating your own module and don't plan to integrate it with Godot, you don't need to make a separate pull request for your icons to be available within the editor as they can be self-contained.

For specific instructions on how to create module icons, refer to Creating custom module icons.

Troubleshooting

If icons don't appear in the editor, make sure that:

1. Each icon's filename matches the naming requirement as described previously.
2. `modules/svg` is enabled (it should be enabled by default). Without it, icons won't appear in the editor at all.

References

- editor/icons [https://github.com/godotengine/godot/tree/master/editor/icons]

Editor style guide

Introduction

Thanks for your interest in contributing to the Godot editor!

This page describes the grammar and writing style used throughout the Godot editor. Following this style guide will help your contribution get merged faster since there will be fewer review steps required.

Writing style

- **Write messages (errors, warnings, ...) as full sentences.** They should start with an uppercase letter and end with a period.
- **Try to keep sentences short.** This makes it more likely that their translations will be short as well, which is a good thing to avoid UI bugs.
- **Use contractions.** For example, use "isn't" instead of "is not". An exception to this rule can be made when you specifically want to emphasize one of the contraction's words.
- **Use double quotes in messages** (`""`) instead of single quotes (`''`). Double quotes should be used to quote user input, file paths and possibly other things depending on the context.

See also

Try to follow the Writing guidelines in addition to the guidelines outlined above.

Button and menu texts

Capitalize text in buttons and menu actions:

- **Good:** *Open Editor Data Folder*
- **Bad:** *Open editor data folder*

If a menu action opens a modal dialog, suffix it with an ellipsis (`...`).

- **Good:** *Editor Settings...*
- **Bad:** *Editor Settings*

Inspector sections

In general, don't create sections that contain less than 3 items. Sections that contain few items make it difficult to navigate the inspector, while missing the benefits of using sections such as folding.

There are some valid exceptions for this, such as material features in StandardMaterial3D.

This advice also applies to the Project Settings and Editor Settings.

Inspector performance hints

Enum properties that noticeably impact performance should have a performance hint associated. The hint should refer to the *absolute* performance impact, rather than being relative to the other options provided in the enum. Here are some examples taken from the Godot editor:

- **Screen-space antialiasing:** *Disabled (Fastest), FXAA (Fast)*

- **MSAA quality:** *Disabled (Fastest), 2x (Fast), 4x (Average), 8x (Slow), 16x (Slower)*
- **DirectionalLight mode:** *Orthogonal (Fast), PSSM 2 Splits (Average), PSSM 4 Splits (Slow)*

For consistency, try to stick to the terms below (from fastest to slowest):

- *Fastest, Faster, Fast, Average, Slow, Slower, Slowest*

Their usage doesn't have to be contiguous. For example, you can use only "Fast" and "Slow" from the list above.

If you're adding a new enum, its values should be ordered from the fastest option to the slowest option.

Tooltips

Consider adding tooltips whenever the action performed by a button or menu action isn't obvious. You can also provide additional context or highlight caveats in the tooltip.

You can do this by calling `set_tooltip(TTR("Text here."))` on the Control-based node in question. If the tooltip is particularly long (more than ~80 characters), wrap it over several lines by adding line breaks using `\n`.

Tooltips should follow the writing style described above. In addition to this, use indicative mood instead of imperative mood:

- **Good:** *Computes global illumination for the selected GIProbe.*
- **Bad:** *Compute global illumination for the selected GIProbe.*

Writing documentation

We always need help to improve the documentation, be it the class reference or the manual. Below, you can find our content and writing guidelines and concrete guides to make changes to the documentation.

Be sure to also check the workflow guidelines, especially if you're new to using Git or GitHub.

Guidelines

Here are the principles and guidelines we strive to follow to write accessible documentation.

- Content guidelines
- Writing guidelines
- Creating documentation images and videos

Writing the manual

These articles explain how to contribute to this very documentation, and how to build the online version locally for testing.

- Contributing to the documentation
- Building the manual with Sphinx

Class reference guides

The pages below focus on the class reference.

As the reference is included in the Godot editor, its source files are part of the godot repository [https://github.com/godotengine/godot]. We use XML files to write it, so the process to contribute to the class reference differs from writing the online manual.

- Contributing to the class reference
- Class reference primer

Translating the documentation

The community is always working hard on making Godot and its documentation available to more people. Localizing the documentation is a colossal and ongoing effort you can be part of.

- Editor and documentation localization

Content guidelines

This document outlines what should be included in the official documentation. Below, you will find a couple of principles and recommendations for writing accessible content.

We want to achieve two goals:

1. **Empathize with our users.** We should write in a way that makes it easy for them to learn from the docs.
2. **Write a complete reference manual**. Our goal here is not to teach programming fundamentals. Instead, our goal is to provide a reference for how Godot's features work.

Guidelines and principles

Below are the guidelines we should strive to follow. They are not hard rules, though: sometimes, a topic will require breaking one or more of them. Still, we should strive to achieve the two goals listed above.

Writing complete and accessible documentation

A feature doesn't exist unless it is documented. If a user can't find information about a feature and how it works, it doesn't exist to them. We should ensure that we cover everything Godot does.

Note

When adding or updating an engine feature, the documentation team needs to know about it. Contributors should open an issue on the *godot-docs* repository when their work gets merged and requires documentation.

Do your best to keep documents **under 1000 words in length**. If a page goes past that threshold, consider splitting it into two parts. Limiting page size forces us to write concisely and to break up large documents so that each page focuses on a particular problem.

Each page or section of a page should clearly state what **problem** it tackles and what it will teach the user. Users need to know if they're reading the correct guide for solving the problems they're encountering. For example, instead of writing the heading "Signals", consider writing "Reacting to changes with signals". The second title makes it clear what the purpose of signals is.

Note

Long section titles lead to long entries in the side menu, which can make navigation cumbersome. Try to keep headings five words long or less.

If the page assumes specific knowledge of other Godot features, mention it and link to the corresponding documentation. For instance, a page about physics may use signals, in which case you could note that the signals tutorial is a prerequisite. You may also link to other websites for prerequisites beyond the documentation's scope. For example, you could link to an introduction to programming in the getting started guide, or a website that teaches math theory in the math section.

Limiting cognitive load

Limit the cognitive load required to read the documentation. The simpler and more explicit language we use, the more efficient it becomes for people to learn. You can do so by:

1. Introducing only one new concept at a time whenever possible.
2. Using simple English, as we recommend in our writing guidelines.
3. Including one or more **concrete usage examples**. Prefer a real-world example to one that uses names like `foo`, `bar`, or `baz`.

While many people may understand more complex language and abstract examples, you will lose others. Understandable writing and practical examples benefit everyone.

Always make an effort to **put yourself in the user's shoes**. When we understand something thoroughly, it becomes obvious to us. We may fail to think about details relevant to a newcomer, but **good documentation meets users where they are**. We should explain each feature's capabilities or intended uses with the most straightforward language possible.

Try to remember what you first needed to know when learning about the feature or concept. What new terms did you need to learn? What confused you? What was the hardest to grasp? You will want users to review your work, and we recommend you practice explaining the feature before writing about it.

Note

Programming fundamentals are a prerequisite for using a complex engine like Godot. Talking about variables, functions, or classes is acceptable. But we should favor plain language over specific terminology like "metaprogramming". If you need to use precise terms, be sure to define them.

Writing guidelines

The Godot community is rich and international. Users come from all around the world. Some of them are young, and many aren't native English speakers. That's why we must all write using a clear and a common language. For the class reference, the goal is to make it easy to read for everyone and precise.

In summary, always try to:

1. Use the active voice
2. Use precise action verbs
3. Avoid verbs that end in -ing
4. Remove unnecessary adverbs and adjectives.
5. Ban these 8 words: obvious, simple, basic, easy, actual, just, clear, and however
6. Use explicit references
7. Use 's to show possession
8. Use the Oxford comma

There are 3 rules to describe classes:

1. Give an overview of the node in the brief description
2. Mention what methods return if it's useful
3. Use "if true" to describe booleans

Note

A technical writer's job is to pack as much information as possible into the smallest and clearest sentences possible. These guidelines will help you work towards that goal.

See also

See the content guidelines for information on the types of documentation you can write in the official documentation.

7 rules for clear English

Use the active voice

Use the active voice when possible. Take the classes, methods, and constants you describe as the subject. It's natural to write using the passive voice, but it's harder to read and produces longer sentences.

Passive:

```
The man **was bitten** by the dog.
```

Active:

```
The dog bit the man.
```

Don't use the passive voice:

```
void edit_set_pivot ( Vector2 pivot )
[...] This method **is implemented** only in some nodes that
inherit Node2D.
```

Do use the node's name as a noun:

```
void edit_set_pivot ( Vector2 pivot )
[...] Only some Node2Ds **implement** this method.
```

Use precise action verbs

Favor precise yet common verbs over generic ones like `make`, `set`, and any expression you can replace with a single word.

Don't repeat the method's name. It already states it sets the pivot value to a new one:

```
void edit_set_pivot ( Vector2 pivot )
Set the pivot position of the 2D node to [code]pivot[/code]
value. [...]
```

Do explain what's the consequence of this "set": use precise verbs like `place`, `position`, `rotate`, `fade`, etc.

```
void edit_set_pivot ( Vector2 pivot )
Position the node's pivot to the [code]pivot[/code] value.
[...]
```

Avoid verbs that end in -ing

The progressive forms describe continuous actions. E.g. "is calling", "is moving".

Don't use the progressive form for instant changes.

```
Vector2 move ( Vector2 rel_vec )
Move the body in the given direction, **stopping** if there is
an obstacle. [...]
```

Do use simple present, past, or future.

```
Vector2 move ( Vector2 rel_vec )
Moves the body in the vector's direction. The body **stops** if
it collides with an obstacle. [...]
```

Exception: If the subject is not clear, replacing "ing" verbs is not an improvement. For example, in the previous sentence, "it replaces" would not make much sense where "replacing" currently is.

You may use the progressive tense to describe actions that are continuous in time. Anything like animation or coroutines.

Tip

Verbs can turn into adjectival nouns with -ing. This is not a conjugation, so you may use them: `the remaining movement`, `the missing file`, etc.

Remove unnecessary adverbs and adjectives

Write as few adjectives and adverbs as possible. Only use them if they add key information to the description.

Don't use redundant or meaningless adverbs. Words that lengthen the documentation but don't add any information:

```
**Basically** a big texture [...]
```

Do write short sentences in a simple, descriptive language:

```
A big texture [...]
```

Ban these 8 words

Don't ever use these 8 banned words:

1. obvious
2. simple
3. basic
4. easy
5. actual
6. just
7. clear
8. however (some uses)

Game creation and programming aren't simple, and nothing's easy to someone learning to use the API for the first time. Other words in the list, like `just` or `actual` won't add any info to the sentence. Don't use corresponding adverbs either: obviously, simply, basically, easily, actually, clearly.

Don't example. The banned words lengthen the description and take attention away from the most important info:

```
**TextureRect**
Control frame that **simply** draws an assigned texture. It can
stretch or not. It's a **simple** way to **just** show an image
in a UI.
```

Do remove them:

```
**TextureRect**
[Control] node that displays a texture. The texture can stretch
to the node's bounding box or stay in the center. Useful to
display sprites in your UIs.
```

"Simple" never helps. Remember, for other users, anything could be complex or frustrate them. There's nothing like a good old *it's simple* to make you cringe. Here's the old brief description, the first sentence on the Timer node's page:

```
**Timer**
A **simple** Timer node.
```

Do explain what the node does instead:

```
**Timer**
Calls a function of your choice after a certain duration.
```

Don't use "basic", it is too vague:

```
**Vector3**
Vector class, which performs **basic** 3D vector math
operations.
```

Do use the brief description to offer an overview of the node:

```
**Vector3**
Provides essential math functions to manipulate 3D vectors:
cross product, normalize, rotate, etc.
```

Use explicit references

Favor explicit references over implicit ones.

Don't use words like "the former", "the latter", etc. They're not the most common in English, and they require you to check the reference.

```
[code]w[/code] and [code]h[/code] define right and bottom
margins. The **latter** two resize the texture so it fits in
the defined margin.
```

Do repeat words. They remove all ambiguity:

```
[code]w[/code] and [code]h[/code] define right and bottom
margins. **[code]w[/code] and [code]h[/code]** resize the
texture so it fits the margin.
```

If you need to repeat the same variable name 3 or 4 times, you probably need to rephrase your description.

Use 's to show possession

Avoid "The milk **of** the cow". It feels unnatural in English. Write "The cow's milk" instead.

Don't write "of the X":

```
The region **of the AtlasTexture that is** used.
```

Do use `'s`. It lets you put the main subject at the start of the sentence, and keep it short:

```
The **AtlasTexture's** used region.
```

Use the Oxford comma to enumerate anything

From the Oxford dictionary:

> The 'Oxford comma' is an optional comma before the word 'and' at the end of a list: *We sell books, videos, and magazines.*
>
> [...] Not all writers and publishers use it, but it can clarify the meaning of a sentence when the items in a list are not single words: *These items are available in black and white, red and yellow, and blue and green.*

Don't leave the last element of a list without a comma:

```
Create a CharacterBody2D node, a CollisionShape2D node and a
sprite node.
```

Do add a comma before *and* or *or*, for the last element of a list with more than two elements.

```
Create a CharacterBody2D node, a CollisionShape2D node, and a
sprite node.
```

How to write methods and classes

Dynamic vs static typing

The code examples in the documentation should follow a consistent style not to confuse users. As static type hints are an optional feature of GDScript, we chose to stick to writing dynamic code. This leads to writing GDScript that is concise and accessible.

The exception is topics that explain static typing concepts to users.

Don't add a type hint with a colon or by casting:

```
const MainAttack := preload("res://fire_attack.gd")
var hit_points := 5
var name: String = "Bob"
var body_sprite := $Sprite2D as Sprite2D
```

Do write constants and variables with dynamic typing:

```
const MainAttack = preload("res://fire_attack.gd")
var hit_points = 5
var name = "Bob"
var body_sprite = $Sprite2D
```

Don't write functions with inferred arguments or return types:

```
func choose(arguments: PackedStringArray) -> String:
    # Chooses one of the arguments from array with equal
chances
    randomize()
    var size := arguments.size()
    var choice: int = randi() % size
    return arguments[choice]
```

Do write functions using dynamic typing:

```
func choose(arguments):
    # Chooses one of the arguments from array with equal
chances
    randomize()
    var size = arguments.size()
    var choice = randi() % size
    return arguments[choice]
```

Use real-world code examples where appropriate

Real-world examples are more accessible to beginners than abstract `foos` and `bars`. You can also copy them directly from your game projects, ensuring that any code snippet compiles without errors.

Writing `var speed = 10` rather than `var my_var = 10` allows beginners to understand code better. It gives them a frame of reference as to where they could use the code snippets in a live project.

Don't write made-up examples:

```
@onready var a = preload("res://MyPath")
@onready var my_node = $MyNode


func foo():
    # Do stuff
```

Do write concrete examples:

```
@onready var sfx_player_gun =
preload("res://Assets/Sound/SFXPlayerGun.ogg")
@onready var audio_player = $Audio/AudioStreamPlayer


func play_shooting_sound():
    audio_player.stream = sfx_player_gun
    audio_player.play()
```

Of course, there are times when using real-world examples is impractical. In those situations, you should still avoid using names such as `my_var`, `foo()` or `my_func()` and consider more meaningful names for your examples.

Give an overview of the node in the brief description

The brief description is the reference's most important sentence. It's the user's first contact with a node:

1. It's the only description in the "Create New Node" dialog.
2. It's at the top of every page in the reference

The brief description should explain the node's role and its functionality, in up to 200 characters.

Don't write tiny and vague summaries:

```
**Node2D**
Base node for 2D system.
```

Do give an overview of the node's functionality:

```
**Node2D**
A 2D game object, inherited by all 2D-related nodes. Has a
position, rotation, scale, and Z index.
```

Use the node's full description to provide more information, and a code example, if possible.

Mention what methods return if it's useful

Some methods return important values. Describe them at the end of the description, ideally on a new line. No need to mention the return values for any method whose name starts with `set` or `get`.

Don't use the passive voice:

```
Vector2 move ( Vector2 rel_vec )
[...] The returned vector is how much movement was remaining
before being stopped.
```

Do always use "Returns".

```
Vector2 move ( Vector2 rel_vec )
[...] Returns the remaining movement before the body was
stopped.
```

Notice the exception to the "direct voice" rule: with the move method, an external collider can influence the method and the body that calls `move`. In this case, you can use the passive voice.

Use "if true" to describe booleans

For boolean member variables, always use `if true` and/or `if false`, to stay explicit. `Controls whether or not` may be ambiguous and won't work for every member variable.

Also, surround boolean values, variable names and methods with `[code][/code]`.

Do start with "if true":

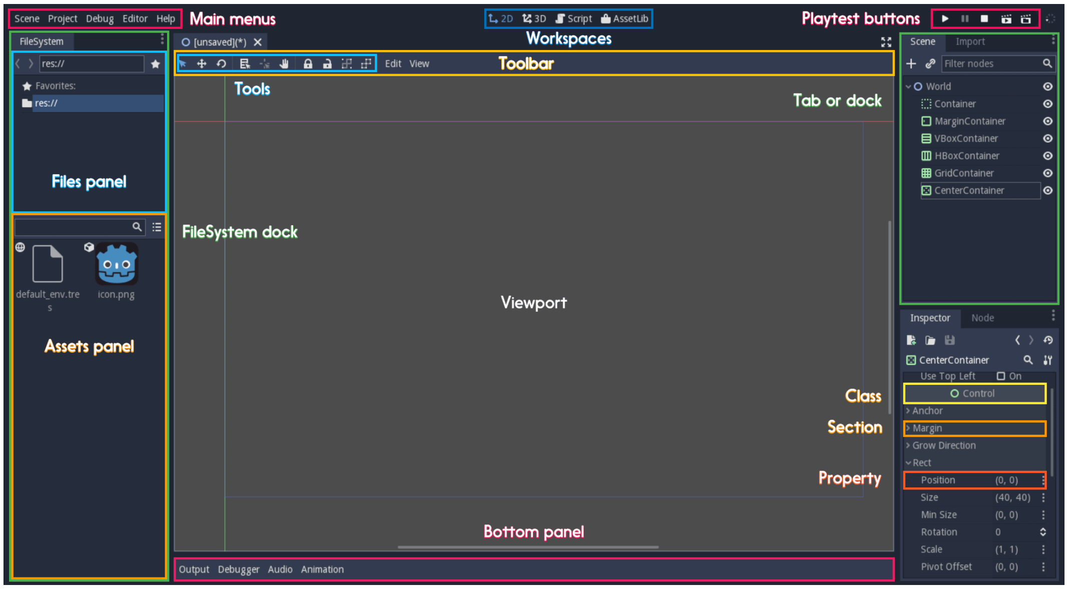
```
Timer.autostart
If [code]true[/code], the timer will automatically start when
entering the scene tree.
```

Use `[code]` around arguments

In the class reference, always surround arguments with `[code][/code]`. In the documentation and in Godot, it will display like `this`. When you edit XML files in the Godot repository, replace existing arguments written like 'this' or \`this\` with `[code]this[/code]`.

Common vocabulary to use in Godot's documentation

The developers chose some specific words to refer to areas of the interface. They're used in the sources, in the documentation, and you should always use them instead of synonyms, so the users know what you're talking about.


Main menus
Workspaces
Playtest buttons
Toolbar
Tools
Tab or dock
Files panel
FileSystem dock
Viewport
Assets panel
Class
Section
Property
Bottom panel
Scene Project Debug Editor Help
2D 3D Script AssetLib
FileSystem
res://
Favorites:
default_env.tres
icon.png
[unsaved](*)
Edit View
Scene Import
Filter nodes
World
Container
MarginContainer
VBoxContainer
HBoxContainer
GridContainer
CenterContainer
Inspector Node
Use Top Left
On
Control
Anchor
Margin
Grow Direction
Rect
Position (0, 0)
Size (40, 40)
Min Size (0, 0)
Rotation 0
Scale (1, 1)
Pivot Offset (0, 0)
Output Debugger Audio Animation

Overview of the interface and common vocabulary

In the top left corner of the editor lie the `main menus`. In the center, the buttons change the `workspace`. And together the buttons in the top right are the `playtest buttons`. The area in the center, that displays the 2D or the 3D space, is the `viewport`. At its top, you find a list of `tools` inside the `toolbar`.

The tabs or dockable panels on either side of the viewport are `docks`. You have the `FileSystem dock`, the `Scene dock` that contains your scene tree, the `Import dock`, the `Node dock`, and the `Inspector` or `Inspector dock`. With the default layout you may call the tabbed docks `tabs`: the `Scene tab`, the `Node tab`...

The Animation, Debugger, etc. at the bottom of the viewport are `panels`. Together they make up the `bottom panels`.

Foldable areas of the Inspector are `sections`. The node's parent class names, which you can't fold, are `Classes` e.g. the `CharacterBody2D class`. And individual lines with key-value pairs are `properties`. E.g. `position` or `modulate color` are both `properties`.

Keyboard shortcut guidelines

Keyboard and mouse shortcuts should make use of the `:kbd:` tag, which allows shortcuts to stand out from the rest of the text and inline code. Use the compact form for modifier keys (`Ctrl`/`Cmd`) instead of their spelled out form (`Control`/`Command`). For combinations, use the `+` symbol with a space on either side of the symbol.

Make sure to mention shortcuts that differ on macOS compared to other platforms. You can find a list of all shortcuts, including what they are on macOS, on this page.

Try to integrate the shortcut into sentences the best you can. Here are some examples with the `:kbd:` tag left as-is for better visibility:

- Press ``:kbd:`Ctrl + Alt + T` `` to toggle the panel (``:kbd:`Opt + Cmd + T` `` on macOS).
- Press ``:kbd:`Space` `` and hold the left mouse button to pan in the 2D editor.
- Press ``:kbd:`Shift + Up Arrow` `` to move the node upwards by 8 pixels.

Creating documentation images and videos

Throughout the documentation, images are often needed to make the explanation of a feature or concept as clear as possible for a reader. This page will explain the process from beginning to end.

Images

Capturing an image

To take a picture of something in Godot, a screen capture tool can be used.

On Windows 10 and 11 that would be the Snip & Sketch program. Pressing `Windows + Shift + S` lets you take a screenshot of a portion of the screen and save it to the clipboard. After pressing those keys, click and drag over the area you wish to take a picture of.

On macOS, pressing `Shift + Command + 3` does the same. To take a picture of the entire screen press `Shift + Command + 4`. All screenshots taken will be saved to the desktop.

Each Linux desktop environment has it's own screenshot tool. For example, on KDE Plasma the program Spectacle is used for taking screenshots. If your distribution doesn't come with one by default try searching its package repository, or Flathub if that's supported.

All screenshots should ideally be taken on a 1080p screen. Anything higher resolution is adding detail that doesn't make the documentation better and dramatically increases file size. If you're taking screenshots on a higher resolution screen the screenshot should be scaled down. There are instructions on how to do this later on this page.

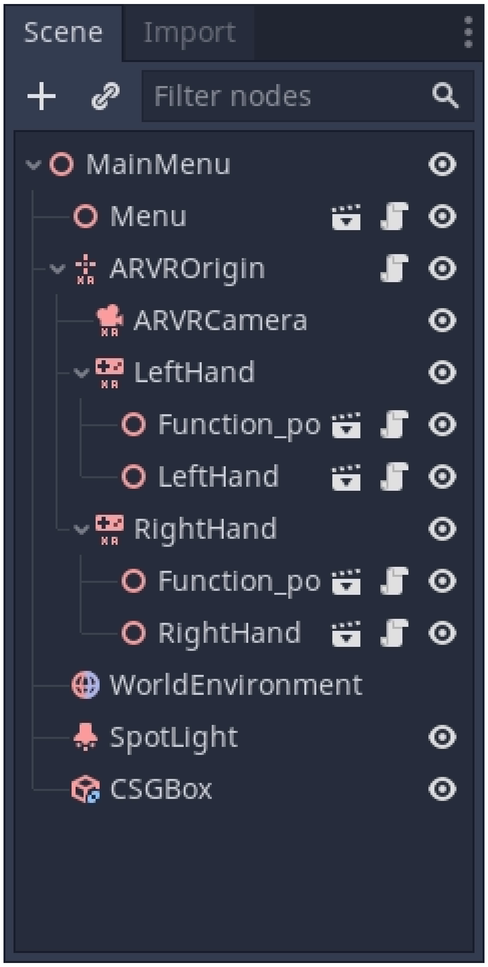
Format conversion

The current format for images in Godot's documentation is WebP (`.webp`). While some Linux programs will support saving screenshots in this format, macOS and the Snip & Sketch program on Windows do not. For images that don't need editing, such as precise cropping or adding outlines, Squoosh can be used. Squoosh [https://squoosh.app/] is a converter developed by Google, is open source, and doesn't give Google any image rights by using it. When choosing compression if you can get an image that's under 300KB in size use lossless compression. If it's over 300KB, use just enough lossy compression to get it under that size. If this results in noticeable compression artifacts using less compression is fine, even if the file size is bigger.

If you already have an image editor such as GIMP, Krita or Photoshop installed it may have the ability to open an image then save it as a WebP file.

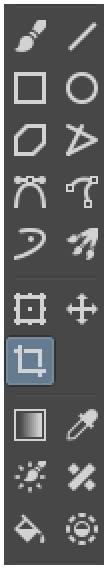
Cropping

For a screenshot of a 2D or 3D scene in the editor, the above steps will be enough. But for most UI images some extra work should be done, specifically cropping to make an image look clean. Below is an example of good cropping.

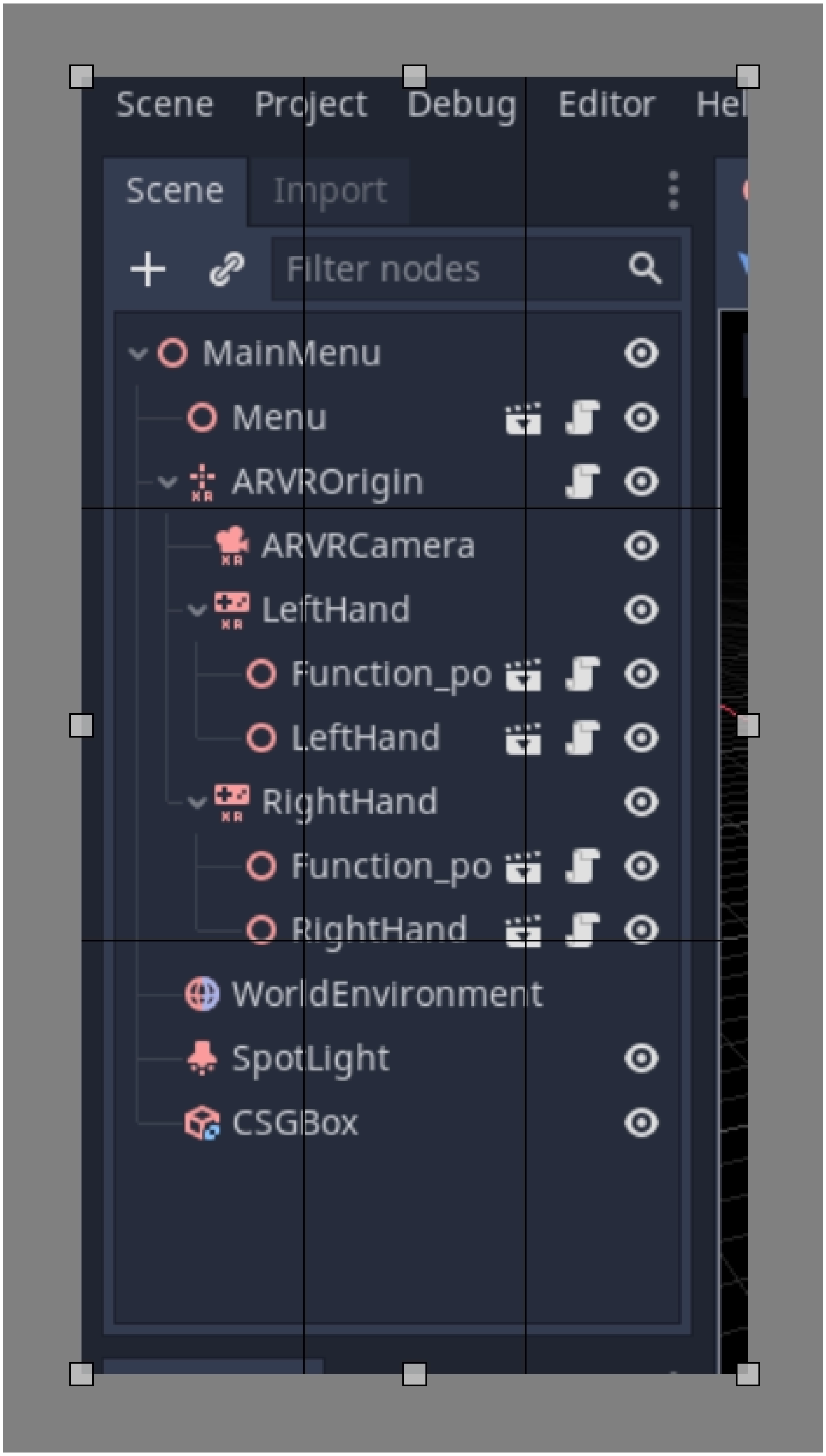

Scene
Import
Filter nodes
MainMenu
Menu
ARVROrigin
ARVRCamera
LeftHand
Function_po
LeftHand
RightHand
Function_po
RightHand
WorldEnvironment
SpotLight
CSGBox

For cropping Krita is the recommended program. While some screenshot programs do have cropping built in it's not always easy to get something precise. And while Krita is designed as a painting program the cropping tool gives you pixel precision by default. Of course, feel free to use a different program you are familiar with.

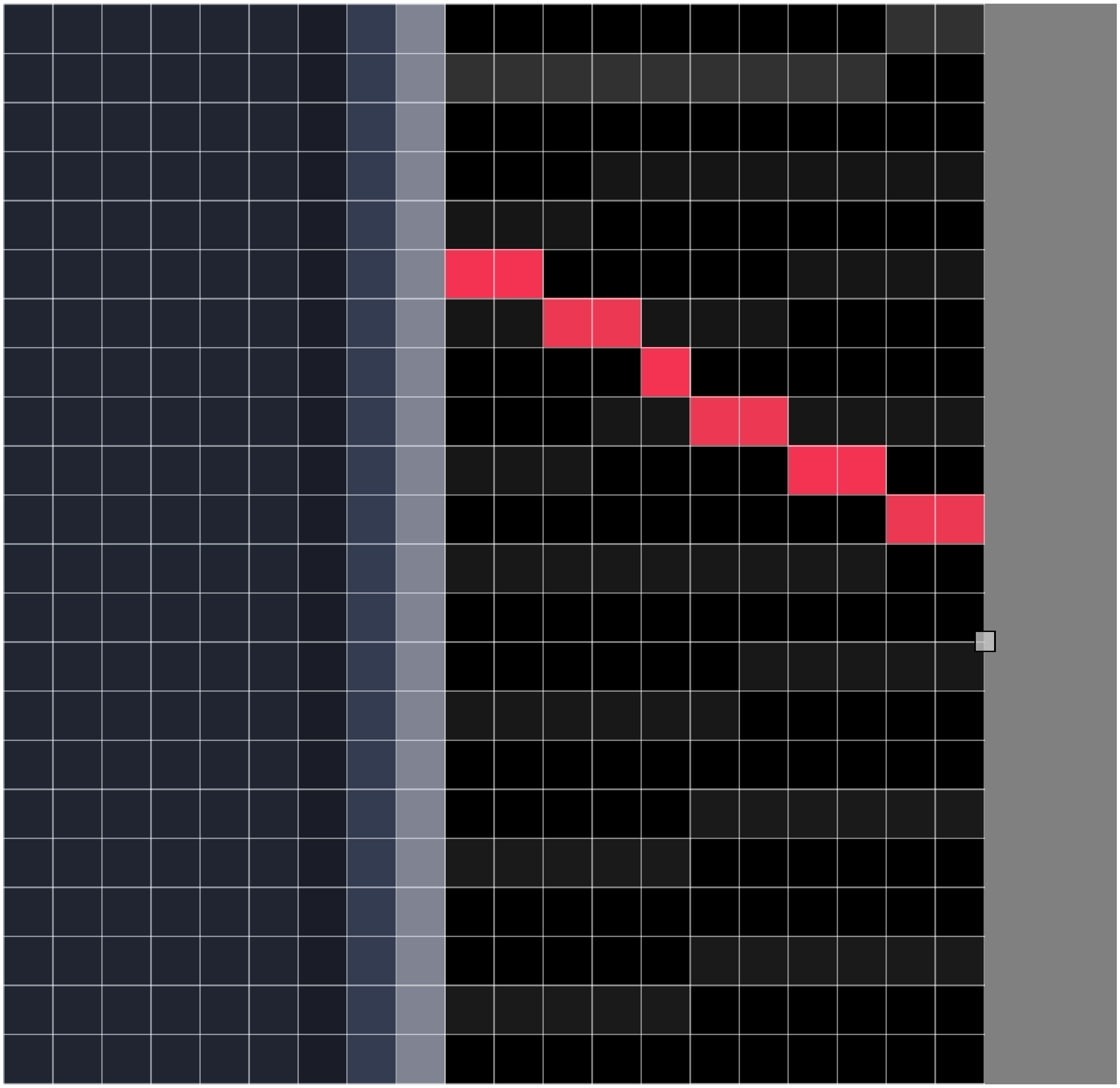
If you've never used Krita before download it from the official Krita website [https://krita.org/en/download/], on Linux you may also be able to download it from your distributions repository, flathub is also an option. Once it's installed on your computer open Krita then open the image you want to crop. This button on the left panel is the crop tool.



After selecting it, click on the image, you should now have cropping tools available.


Scene
Project
Debug
Editor
Hel
Scene
Import
Filter nodes
MainMenu
Menu
ARVROrigin
ARVRCamera
LeftHand
Function_po
LeftHand
RightHand
Function_po
RightHand
WorldEnvironment
SpotLight
CSGBox

Click and drag the white boxes to adjust what gets cropped, if you zoom in close to the image you will see the individual pixels in an image, which is useful for precision.



If you make a mistake and overcrop don't worry, cropping is non-destructive in Krita and can be adjusted. Click on the image with your cropping tool still selected and the controls will return.

Scaling down an image

As explained earlier on this page, all images taken on a screen that is a higher resolution than 1080p should be scaled down. To do this in Krita click on **Image** on the top bar, and from the dropdown menu select **Scale Image To New Size**. This menu can also be opened by pressing `Ctrl + Alt + I`. On this menu you want to adjust the pixel dimensions. For anything taken on a 4K monitor change the value of the width and height to half of its current value, for anything taken on a 1440p monitor multiply the width and height by 0.75. Make sure the **Constrain Proportions** box at the bottom of the menu is checked so you only have to change 1 value.

Saving as WebP in Krita

To save an image as webp if it isn't already one, Go to **File > Save As**. Select **webp** from the **Save as type:** dropdown, then choose wherever you want to save it. After clicking **Save** a menu will popup with webp options. Make sure **Lossless** is checked and **Quality** is set to 100%. This means the image will not lose detail and will be as small as possible.

If the image is over 300KB in size try compressing it losslessly using Squoosh [https://squoosh.app/]. If it's still over 300KB change to lossy compression and slowly increase the compression until it's under 300KB. If this results in noticeable compression artifacts using less compression is fine, even if the file size is bigger.

Outlines, arrows and text

Sometimes an image needs something extra to properly direct the readers attention, or make something clear. Outlines and arrows can be used for this purpose. For these types of edits Inkscape is the recommended open source program, it can be downloaded from the official Inkscape website [https://inkscape.org/]. Like Krita, if you're on Linux you can also check your distributions repository or get it from Flathub.

A full tutorial on creating outlines is not provided here, we recommend searching for various tutorials on how to use it online. However there are two standards for doc image outlines and arrows. First, the color should be yellow, specifically this hex color: `fffb44` (`fffb44ff` if there is a transparency value like in Inkscape). This color was chosen specifically to make sure color blind people do not have issues reading the documentation, other colors can be used in addition to this yellow if multiple outlines on an image are needed, red should be avoided. The second standard is that all outlines and arrow lines should be 2 pixels wide.

Finally, some images might require text to differentiate multiple parts of an image. There are no strict requirements other than use an easy to read non fancy font. As for color the yellow color from before should also be used, but black or other colors can be used if appropriate. For example, if yellow blends into the image, or if there are multiple outlines in multiple colors.

Adding an image to a documentation page

Once you've finished working on your image, it can be added to the documentation. All images are stored in folders named `img` next to the page they are used in.

To add your image, add it to the `img` folder that's in the same folder as the `.rst` file for the page (create it if it doesn't exist). In the `.rst` page, images should be included with the following code snippet:

```
.. image:: img/documentation_image.png
```

Where `documentation_image.webp` would be changed to the name of the image you created. Name your images in a way that makes their meaning clear, possibly with a prefix that makes their relationship to a documentation page explicit.

Videos

Capturing a video

To record a video of something in Godot, a screen capture tool can be used. Operating systems generally don't come with tools that are flexible enough for this, so you'll need to install a third-party utility.

OBS Studio [https://obsproject.com/] is the most popular option, but SimpleScreenRecorder [https://www.maartenbaert.be/simplescreenrecorder/] can be used as an alternative on Linux. ShareX [https://getsharex.com/] can be used as an alternative on Windows. All these tools can be configured to record the entire screen, a specific window or a predetermined rectangle.

The recommended framerate for video recordings is 60 FPS, although you can use 30 FPS for longer videos to reduce their file size. For fullscreen videos, use a resolution of 1280×720.

Note

Godot's Movie Maker mode can be used to record the output of a running project, including its audio. This doesn't require installing any third-party software and avoids any frame drops (even when recording on a slow device), but it's less flexible.

Compressing the captured video

The recommendation is to record your video in the highest quality possible (without dropping frames due to excessive CPU/GPU utilization), then re-encode it later to reduce its file size. This results in more efficient compression than directly aiming for a small file size, as real-time compression methods are less efficient than slower compression methods.

To re-encode videos for a smaller file size, use HandBrake [https://handbrake.fr/] or the *FFmpeg <https://ffmpeg.org/>* command line below:

```
ffmpeg -i input.mp4 -crf 23 output.webm
```

The number after `-crf` adjusts the video quality, with higher numbers resulting in *lower* quality (and smaller file sizes). A CRF of `23` is a good starting point, but you may need to use a higher value for longer videos to ensure the file size remains reasonable. Try to aim for a file size under 2 MB if possible.

If the video was recorded in a higher resolution or framerate, you can adjust its output resolution and framerate as follows:

```
ffmpeg -i input.mp4 -crf 23 -vf scale=1280:-2 -r 30 output.webm
```

This results in a video resolution around 1280×720 at 30 FPS. The exact video resolution will vary depending on the source's aspect ratio.

Tip

If the video was recorded with an audio track but this audio track is not necessary, consider stripping it by adding the `-an` option to the FFmpeg command line (before the output file name). This will reduce file size and also ensure audio controls don't show up on the video when played in a browser.

Adding a video to a documentation page

Once you've finished working on your video, it can be added to the documentation. All videos are stored in folders named `video` next to the page they are used in.

To add your video, add it to the `video` folder that's in the same folder as the `.rst` file for the page (create it if it doesn't exist). In the `.rst` page, videos should be included with the following code snippet:

```
.. video:: video/csg_tools.webm
   :alt: Put a text description of the video here
   :autoplay:
   :loop:
   :muted:
```

Where `documentation_video.webp` would be changed to the name of the video you created. Name your videos in a way that makes their meaning clear, possibly with a prefix that makes their relationship to a documentation page explicit.

The `:autoplay:`, `:loop:` and `:muted:` flags should always be specified unless the video needs to play audio. In this case, do not specify *any* of these flags.

Contributing to the documentation

This guide explains how to contribute to Godot's documentation, be it by writing or reviewing pages.

See also

If you want to translate pages or the class reference from English to other languages, read Editor and documentation localization.

Getting started

To modify or create pages in the reference manual, you need to edit `.rst` files in the godot-docs GitHub repository [https://github.com/godotengine/godot-docs]. Modifying those pages in a pull request triggers a rebuild of the online documentation upon merging.

See also

For details on Git usage and the pull request workflow, please refer to the Pull request workflow page. Most of what it describes regarding the main godotengine/godot repository is also valid for the docs repository.

Warning

The class reference's source files are in the Godot engine repository [https://github.com/godotengine/godot]. We generate the

Class Reference section of this documentation from them. If you want to update the description of a class, its methods, or properties, read Contributing to the class reference.

What is the Godot documentation

The Godot documentation is intended as a comprehensive reference manual for the Godot game engine. It is not meant to contain step-by-step tutorials, except for two game creation tutorials in the Getting Started section.

We strive to write factual content in an accessible and well-written language. To contribute, you should also read:

1. Writing guidelines. There, you will find rules and recommendations to write in a way that everyone understands.
2. Content guidelines. They explain the principles we follow to write the documentation and the kind of content we accept.

Contributing changes

Pull Requests should use the `master` branch by default. Only make Pull Requests against other branches (e.g. `2.1` or `3.0`) if your changes only apply to that specific version of Godot.

Though less convenient to edit than a wiki, this Git repository is where we write the documentation. Having direct access to the source files in a revision control system is a plus to ensure our documentation quality.

Editing existing pages

To edit an existing page, locate its `.rst` source file and open it in your favorite text editor. You can then commit the changes, push them to your fork, and make a pull request. **Note that the pages in `classes/` should not be edited here.** They are automatically generated from Godot's XML class reference [https://github.com/godotengine/godot/tree/master/doc/classes]. See Contributing to the class reference for details.

See also

To build the manual and test changes on your computer, see Building the manual with Sphinx.

Editing pages online

You can edit the documentation online by clicking the **Edit on GitHub** link in the top-right of every page.

Doing so takes you to the GitHub text editor. You need to have a GitHub account and to log in to use it. Once logged in, you can propose change like so:

1. Click the **Edit on GitHub** button.
2. On the GitHub page you're taken to, click the pencil icon in the top-right corner near the **Raw**, **Blame**, and **Delete** buttons. It has the tooltip "Fork this project and edit the file".
3. Edit the text in the text editor.
4. At the bottom of the web page, summarize the changes you made and click the button **Propose file change**. Make sure to replace the placeholder "Update file.rst" by a short but clear one-line description, as this is the commit title.
5. On the following screens, click the **Create pull request** button until you see a message like *Username wants to merge 1 commit into godotengine:master from Username:patch-1*.

Another contributor will review your changes and merge them into the docs if they're good. They may also make changes or ask you to do so before merging.

Adding new pages

Before adding a new page, please ensure that it fits in the documentation:

1. Look for existing issues [https://github.com/godotengine/godot-docs/issues] or open a new one to see if the page is necessary.
2. Ensure there isn't a page that already covers the topic.
3. Read our Content guidelines.

To add a new page, create a `.rst` file with a meaningful name in the section you want to add a file to, e.g. `tutorials/3d/light_baking.rst`.

You should then add your page to the relevant "toctree" (table of contents, e.g. `tutorials/3d/index.rst`). Add your new filename to the list on a new line, using a relative path and no extension, e.g. here `light_baking`.

Titles

Always begin pages with their title and a Sphinx reference name:

```
.. _doc_insert_your_title_here:

Insert your title here
======================
```

The reference `_doc_insert_your_title_here` and the title should match.

The reference allows linking to this page using the `:ref:` format, e.g. ``:ref:`doc_insert_your_title_here` `` would link to the above example page (note the lack of leading underscore in the reference).

Write your titles like plain sentences, without capitalizing each word:

- **Good:** Understanding signals in Godot
- **Bad:** Understanding Signals In Godot

Only proper nouns, projects, people, and node class names should have their first letter capitalized.

Sphinx and reStructuredText syntax

Check Sphinx's reST Primer [https://www.sphinx-doc.org/en/stable/rest.html] and the official reference [https://docutils.sourceforge.net/rst.html] for details on the syntax.

Sphinx uses specific reST comments to do specific operations, like defining the table of contents (`.. toctree::`) or cross-referencing pages. Check the official Sphinx documentation [https://www.sphinx-doc.org/en/stable/index.html] for more details. To learn how to use Sphinx directives like `.. note::` or `.. seealso::`, check out the Sphinx directives documentation [https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html].

Adding images and attachments

To add images, please put them in an `img/` folder next to the `.rst` file with a meaningful name and include them in your page with:

```
.. image:: img/image_name.png
```

Alternatively, you can use the *figure* directive, which gives the image a contrasting border and allows centering it on the page.

```
.. figure:: img/image_name.png
    :align: center
```

You can also include attachments as support material for a tutorial, by placing them into a `files/` folder next to the `.rst` file, and using this inline markup:

```
:download:`file_name.zip <files/file_name.zip>`
```

Consider using the *godot-docs-project-starters <https://github.com/godotengine/godot-docs-project-starters>* repository for hosting support materials, such as project templates and asset packs. You can use a direct link to the generated archive from that repository with the regular link markup:

```
`file_name.zip <https://github.com/godotengine/godot-docs-project-starters/releases/download/latest-4.x/file_name.zip>`_
```

License

This documentation and every page it contains is published under the terms of the Creative Commons Attribution 3.0 license (CC BY 3.0) [https://creativecommons.org/licenses/by/3.0/], with attribution to "Juan Linietsky, Ariel Manzur and the Godot community".

By contributing to the documentation on the GitHub repository, you agree that your changes are distributed under this license.

Building the manual with Sphinx

This page explains how to build a local copy of the Godot manual using the Sphinx docs engine. This allows you to have local HTML files and build the documentation as a PDF, EPUB, or LaTeX file, for example.

Before you get started, make sure that you have:

- Git [https://git-scm.com/]
- make [https://www.gnu.org/software/make/] (unless you're using Windows)
- Python [https://www.python.org/] 3

Note

Python 3 should come with the `pip3` command. You may need to write `python3 -m pip` (Unix) or `py -m pip` (Windows) instead of `pip3`. If both approaches fail, make sure that you have pip3 installed [https://pip.pypa.io/en/stable/installation/].

1. *(Optional)* Set up a virtual environment. Virtual environments prevent potential conflicts between the Python packages in `requirements.txt` and other Python packages that are installed on your system.

   a. Create the virtual environment:

      Windows

```
py -m venv godot-docs-venv
```

Other platforms

```
python3 -m venv godot-docs-venv
```

b. Activate the virtual environment:

Windows

```
godot-docs-venv\Scripts\activate.bat
```

Other platforms

```
source godot-docs-venv/bin/activate
```

c. *(Optional)* Update pre-installed packages:

Windows

```
py -m pip install --upgrade pip setuptools
```

Other platforms

```
pip3 install --upgrade pip setuptools
```

2. Clone the docs repo:

```
git clone https://github.com/godotengine/godot-docs.git
```

3. Change directory into the docs repo:

```
cd godot-docs
```

4. Install the required packages:

```
pip3 install -r requirements.txt
```

5. Build the docs:

```
make html
```

Note

On Windows, that command will run `make.bat` instead of GNU Make (or an alternative).

Alternatively, you can build the documentation by running the sphinx-build program manually:

```
sphinx-build -b html ./ _build/html
```

The compilation will take some time as the `classes/` folder contains hundreds of files. See Hints for performance.

You can then browse the documentation by opening `_build/html/index.html` in your web browser.

Dealing with errors

If you run into errors, you may try the following command:

```
make SPHINXBUILD=~/.local/bin/sphinx-build html
```

If you get a `MemoryError` or `EOFError`, you can remove the `classes/` folder and run `make` again. This will drop the class references from the final HTML documentation but will keep the rest intact.

Important

If you delete the `classes/` folder, do not use `git add .` when working on a pull request or the whole `classes/` folder will be removed when you commit. See #3157 [https://github.com/godotengine/godot-docs/issues/3157] for more detail.

Hints for performance

RAM usage

Building the documentation requires at least 8 GB of RAM to run without disk swapping, which slows it down. If you have at least 16 GB of RAM, you can speed up compilation by running:

Windows

```
set SPHINXOPTS=-j2 && make html
```

Other platforms

```
make html SPHINXOPTS=-j2
```

You can use `-j auto` to use all available CPU threads, but this can use a lot of RAM if you have a lot of CPU threads. For instance, on a system with 32 CPU threads, `-j auto` (which corresponds to `-j 32` here) can require 20+ GB of RAM for Sphinx alone.

Specifying a list of files

You can specify a list of files to build, which can greatly speed up compilation:

```
make FILELIST='classes/class_node.rst
classes/class_resource.rst' html
```

Contributing to the class reference

The Class reference is a set of articles describing the public API of the engine. This includes descriptions for various classes, methods, properties, and global objects, available for scripting. The class reference is available online, from the documentation sidebar, and in the Godot editor, from the help menu.

As the engine grows and features are added or modified, some parts of the class reference become obsolete and new descriptions and examples need to be added. While developers are encouraged to document all of their work in the class reference when submitting a pull request, we can't expect everyone to be able to write high quality documentation, so there is always work for contributors like you to polish existing and create missing reference material.

The source of the class reference

As the class reference is available in two places, online and in the editor, we need to take care to keep things in sync. To achieve this the main Godot repository [https://github.com/godotengine/godot/] is picked as the source of truth, and the documentation for the class reference is tracked there.

Warning

You should **not** edit `.rst` files in the `classes/` folder of the documentation repository [https://github.com/godotengine/godot-docs/]. These files are generated automatically and are synced manually by project maintainers. Read further to learn how to correctly edit the class reference.

In the main repository the class reference is stored in XML files, one for each exposed class or global object. The majority of these files is located in doc/classes/ [https://github.com/godotengine/godot/tree/master/doc/classes], but some modules contain their own documentation as well. You will find it in the `modules/<module_name>/doc_classes/` directory. To learn more about editing XML files refer to Class reference primer.

See also

For details on Git usage and the pull request workflow, please refer to the Pull request workflow page.

If you want to translate the class reference from English to another language, see Editor and documentation localization. This guide is also available as a video tutorial on YouTube [https://www.youtube.com/watch?v=5jeHXxeX-JY].

Important: If you plan to make large changes, you should create an issue on the godot-docs repository [https://github.com/godotengine/godot-docs/] or comment on an existing issue. Doing so lets others know you're already taking care of a given class.

What to contribute

The natural place to start contributing is the classes that you are most familiar with. This ensures that the added description will be based on experience and the necessary know-how, not just the name of a method or a property. We advise not to add low effort descriptions, no matter how appealing it may look. Such descriptions obscure the need for documentation and are hard to identify automatically.

See also

Following this principle is important and allows us to create tools for contributors. Such as the class reference's completion status tracker [https://godotengine.github.io/doc-status/]. You can use it to quickly find documentation pages missing descriptions.

If you decide to document a class, but don't know what a particular method does, don't worry. Leave it for now, and list the methods you skipped when you open a pull request with your changes. Another writer will take care of it.

You can still look at the methods' implementation in Godot's source code on GitHub. If you have doubts, feel free to ask on the Q&A website [https://ask.godotengine.org/] and Godot Contributors Chat [https://chat.godotengine.org/].

Warning

Unless you make minor changes, like fixing a typo, we do not recommend using the GitHub web editor to edit the class reference's XML files. It lacks features to edit XML well, like keeping indentations consistent, and it does not allow amending commits based on reviews.

It also doesn't allow you to test your changes in the engine or with validation scripts as described in How to edit class XML.

Updating class reference when working on the engine

When you create a new class or modify an existing engine's API, you need to re-generate the XML files in `doc/classes/`.

To do so, you first need to compile Godot. See the Introduction to the buildsystem page to learn how. Then, execute the compiled Godot binary from the Godot root directory with the `--doctool` option. For example, if you're on 64-bit Linux, the command might be:

```
./bin/godot.linuxbsd.editor.x86_64 --doctool
```

The exact set of suffixes may be different. Carefully read through the linked article to learn more about that.

The XML files in `doc/classes/` should then be up-to-date with current Godot Engine features. You can then check what changed using the `git diff` command.

Please only include changes that are relevant to your work on the API in your commits. You can discard changes in other XML files using `git checkout`, but consider reporting if you notice unrelated files being updated. Ideally, running this command should only bring up the changes that you yourself have made.

You will then need to add descriptions to any newly generated entries.

Class reference primer

This page explains how to write the class reference. You will learn where to write new descriptions for the classes, methods, and properties for Godot's built-in node types.

See also

To learn to submit your changes to the Godot project using the Git version control system, see Contributing to the class reference.

The reference for each class is contained in an XML file like the one below:

```
<class name="Node2D" inherits="CanvasItem" version="4.0">
    <brief_description>
        A 2D game object, inherited by all 2D-related nodes. Has a position, rotation, scale, and Z index.
    </brief_description>
    <description>
        A 2D game object, with a transform (position, rotation, and scale). All 2D nodes, including physics objects and sprites, inherit from Node2D. Use Node2D as a parent node to move, scale and rotate children in a 2D project. Also gives control of the node's render order.
    </description>
    <tutorials>
        <link title="Custom drawing in 2D">https://docs.godotengine.org/en/latest/tutorials/2d/custom_drawing_in_2d.html</link>
        <link title="All 2D Demos">https://github.com/godotengine/godot-demo-projects/tree/master/2d</link>
    </tutorials>
    <methods>
        <method name="apply_scale">
            <return type="void">
            </return>
            <argument index="0" name="ratio" type="Vector2">
            </argument>
            <description>
                Multiplies the current scale by the [code]ratio[/code] vector.
            </description>
        </method>
        [...]
        <method name="translate">
            <return type="void">
            </return>
            <argument index="0" name="offset" type="Vector2">
            </argument>
            <description>
                Translates the node by the given [code]offset[/code] in local coordinates.
            </description>
        </method>
    </methods>
    <members>
        <member name="global_position" type="Vector2" setter="set_global_position" getter="get_global_position">
            Global position.
        </member>
        [...]
        <member name="z_index" type="int" setter="set_z_index" getter="get_z_index" default="0">
            Z index. Controls the order in which the nodes render. A node with a higher Z index will display in front of others.
        </member>
    </members>
    <constants>
    </constants>
</class>
```

It starts with brief and long descriptions. In the generated docs, the brief description is always at the top of the page, while the long description lies below the list of methods, variables, and constants. You can find methods, member variables, constants, and signals in separate XML nodes.

For each, you want to learn how they work in Godot's source code. Then, fill their documentation by completing or improving the text in these tags:

- *<brief_description>*
- *<description>*
- *<constant>*
- *<method>* (in its *<description>* tag; return types and arguments don't take separate documentation strings)
- *<member>*
- *<signal>* (in its *<description>* tag; arguments don't take separate documentation strings)
- *<constant>*

Write in a clear and simple language. Always follow the writing guidelines to keep your descriptions short and easy to read. **Do not leave empty lines** in the descriptions: each line in the XML file will result in a new paragraph, even if it is empty.

How to edit class XML

Edit the file for your chosen class in `doc/classes/` to update the class reference. The folder contains an XML file for each class. The XML lists the constants and methods you will find in the class reference. Godot generates and updates the XML automatically.

Note

For some modules in the engine's source code, you'll find the XML files in the `modules/<module_name>/doc_classes/` directory instead.

Edit it using your favorite text editor. If you use a code editor, make sure that it doesn't change the indent style: you should use tabs for the XML and four spaces inside BBCode-style blocks. More on that below.

To check that the modifications you've made are correct in the generated documentation, navigate to the `doc/` folder and run the command `make rst`. This will convert the XML files to the online documentation's format and output errors if anything's wrong.

Alternatively, you can build Godot and open the modified page in the built-in code reference. To learn how to compile the engine, read the compilation guide.

We recommend using a code editor that supports XML files like Vim, Atom, Visual Studio Code, Notepad++, or another to comfortably edit the file. You can also use their search feature to find classes and properties quickly.

Tip

If you use Visual Studio Code, you can install the vscode-xml extension [https://marketplace.visualstudio.com/items?itemName=redhat.vscode-xml] to get linting for class reference XML files.

Improve formatting with BBCode style tags

Godot's XML class reference supports BBCode-like tags for linking as well as formatting text and code. In the tables below you can find the available tags, usage examples and the results after conversion to reStructuredText.

Linking

Whenever you link to a member of another class, you need to specify the class name. For links to the same class, the class name is optional and can be omitted.

Tag and Description	Example	Result
`[Class]` Link to class	`Move the [Sprite2D].`	Move the Sprite2D.
`[annotation Class.name]` Link to annotation	`See [annotation @GDScript.@rpc].`	See @GDScript.@rpc.
`[constant Class.name]` Link to constant	`See [constant Color.RED].`	See Color.RED.
`[enum Class.name]` Link to enum	`See [enum Mesh.ArrayType].`	See Mesh.ArrayType.
`[member Class.name]` Link to member	`Get [member Node2D.scale].`	Get Node2D.scale.
`[method Class.name]` Link to method	`Call [method Node3D.hide].`	Call Node3D.hide().
`[constructor Class.name]` Link to built-in constructor	`Use [constructor Color.Color].`	Use Color.Color.

Tag and Description	Example	Result
`[operator Class.name]` Link to built-in operator	`Use [operator Color.operator *].`	Use Color.operator *.
`[signal Class.name]` Link to signal	`Emit [signal Node.renamed].`	Emit Node.renamed.
`[theme_item Class.name]` Link to theme item	`See [theme_item Label.font].`	See Label.font.
`[param name]` Parameter name (as code)	`Takes [param size] for the size.`	Takes `size` for the size.

Note

Currently only @GDScript has annotations.

Formatting text

Tag and Description	Example	Result
`[br]` Line break	`Line 1.[br]` `Line 2.`	Line 1. Line 2.
`[lb]` `[rb]`	`[lb]b[rb]text[lb]/b[rb]`	[b]text[/b]

Tag and Description	Example	Result
`[` and `]` respectively		
`[b] [/b]` Bold	`Do [b]not[/b] call this method.`	Do **not** call this method.
`[i] [/i]` Italic	`Returns the [i]global[/i] position.`	Returns the *global* position.
`[u] [/u]` Underline	`[u]Always[/u] use this method.`	Always use this method.
`[s] [/s]` Strikethrough	`[s]Outdated information.[/s]`	~~Outdated information.~~
`[url] [/url]` Hyperlink	`[url]https://example.com[/url]` `[url=https://example.com]Website[/url]`	https://example.com Website

`[center]`
`[/center]`
Horizontal centering
	`[center]2 + 2 = 4[/center]`	2 + 2 = 4

`[kbd] [/kbd]` Keyboard/mouse shortcut	`Press [kbd]Ctrl + C[/kbd].`	Press `Ctrl + C`.
`[code] [/code]` Inline code fragment	`Returns [code]true[/code].`	Returns `true`.

Note

1. Some supported tags like `[color]` and `[font]` are not listed here because they are not recommended in the engine documentation.
2. `[kbd]` disables BBCode until the parser encounters `[/kbd]`.
3. `[code]` disables BBCode until the parser encounters `[/code]`.

Formatting code blocks

There are two options for formatting code blocks:

1. Use `[codeblock]` if you want to add an example for a specific language.
2. Use `[codeblocks]`, `[gdscript]`, and `[csharp]` if you want to add the same example for both languages, GDScript and C#.

By default, `[codeblock]` highlights GDScript syntax. You can change it using the `lang` attribute. Currently supported options are:

- `[codeblock lang=text]` disables syntax highlighting;
- `[codeblock lang=gdscript]` highlights GDScript syntax;
- `[codeblock lang=csharp]` highlights C# syntax (only in .NET version).

Note

`[codeblock]` disables BBCode until the parser encounters `[/codeblock]`.

Warning

Use `[codeblock]` for pre-formatted code blocks. Inside `[codeblock]`, always use **four spaces** for indentation. The parser will delete tabs.

For example:

```
[codeblock]
func _ready():
    var sprite = get_node("Sprite2D")
    print(sprite.get_pos())
[/codeblock]
```

Will display as:

```
func _ready():
    var sprite = get_node("Sprite2D")
    print(sprite.get_pos())
```

If you need to have different code version in GDScript and C#, use `[codeblocks]` instead. If you use `[codeblocks]`, you also need to have at least one of the language-specific tags, `[gdscript]` and `[csharp]`.

Always write GDScript code examples first! You can use this [experimental code translation tool](https://github.com/HaSa1002/codetranslator) [https://github.com/HaSa1002/codetranslator] to speed up your workflow.

```
[codeblocks]
[gdscript]
func _ready():
    var sprite = get_node("Sprite2D")
    print(sprite.get_pos())
[/gdscript]
[csharp]
public override void _Ready()
{
    var sprite = GetNode("Sprite2D");
    GD.Print(sprite.GetPos());
}
[/csharp]
[/codeblocks]
```

The above will display as:

GDScript

```
func _ready():
    var sprite = get_node("Sprite2D")
    print(sprite.get_pos())
```

C#

```
public override void _Ready()
{
    var sprite = GetNode("Sprite2D");
    GD.Print(sprite.GetPos());
}
```

Formatting notes and warnings

To denote important information, add a paragraph starting with "[b]Note:[/b]" at the end of the description:

```
[b]Note:[/b] Only available when using the Vulkan renderer.
```

To denote crucial information that could cause security issues or loss of data if not followed carefully, add a paragraph starting with "[b]Warning:[/b]" at the end of the description:

```
[b]Warning:[/b] If this property is set to [code]true[/code], it allows
clients to execute arbitrary code on the server.
```

In all the paragraphs described above, make sure the punctuation is part of the BBCode tags for consistency.

Marking API as deprecated/experimental

To mark an API as deprecated or experimental, you need to add the corresponding XML attribute. The attribute value must be a message explaining why the API is not recommended (BBCode markup is supported) or an empty string (the default message will be used). If an API element is marked as deprecated/experimental, then it is considered documented even if the description is empty.

```
<class name="Parallax2D" inherits="Node2D" experimental="This node is meant
to replace [ParallaxBackground] and [ParallaxLayer]. The implementation may
change in the future." xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="../class.xsd">
    [...]
</class>

<constant name="RESPONSE_USE_PROXY" value="305" enum="ResponseCode"
deprecated="Many clients ignore this response code for security reasons. It
is also deprecated by the HTTP standard.">
    HTTP status code [code]305 Use Proxy[/code].
</constant>

<member name="auto_translate" type="bool" setter="set_auto_translate"
getter="is_auto_translating" deprecated="Use [member
Node.auto_translate_mode] instead.">
    Toggles if any text should automatically change to its translated
version depending on the current locale.
</member>

<method name="get_method_call_mode" qualifiers="const" deprecated="Use
[member AnimationMixer.callback_mode_method] instead.">
    <return type="int" enum="AnimationPlayer.AnimationMethodCallMode" />
    <description>
        Returns the call mode used for "Call Method" tracks.
    </description>
</method>
```

Editor and documentation localization

Godot aims to make game development available to everyone, including people who may not know or be comfortable with English. Therefore, we do our best to make the most important resources available in many languages, thanks to the translation effort of the community.

These resources include:

1. The Godot editor's interface [https://hosted.weblate.org/projects/godot-engine/godot/] (ca. 15,000 words).
2. The online documentation [https://hosted.weblate.org/projects/godot-engine/godot-docs/] (editor manual and tutorials, ca. 300,000 words).
3. The class reference [https://hosted.weblate.org/projects/godot-engine/godot-class-reference/], available both online and in the editor (ca. 200,000 words).

To manage translations, we use the GNU gettext file format (`PO` files), and the open source Weblate [https://weblate.org] web-based localization platform, which allows easy collaboration of many contributors to complete the translation for the various components, and keep them up to date. Click the bold links above to access each resource on Weblate.

This page gives an overview of the general translation workflow on Weblate, and some resource-specific instructions on e.g. how to handle some keywords or the localization of images.

Tip

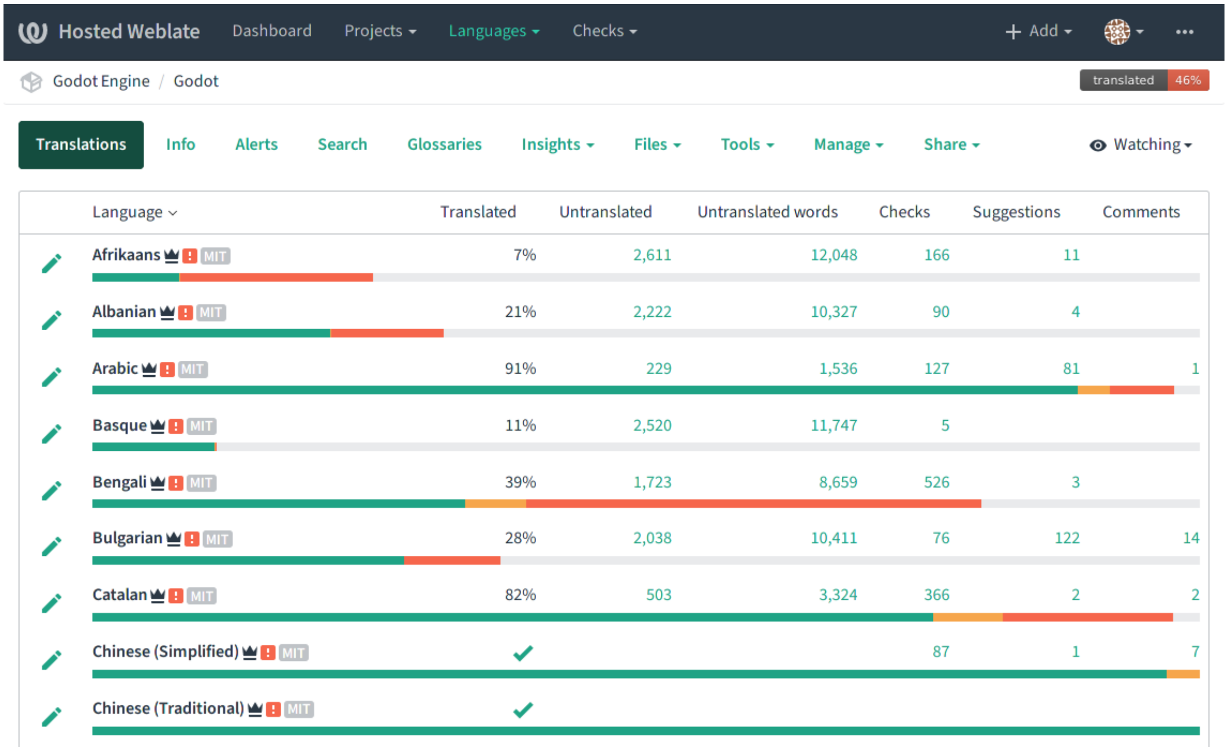
Translating all the official Godot content is a massive undertaking, so we advise prioritizing the resources as they are listed above: first the editor interface, then the online documentation, and eventually the class reference if there are enough translators to keep up with updates.

Using Weblate for translations

While our translations eventually reside in the Git repositories of the Godot engine and its documentation, all translation updates are handled through Weblate, and thus direct pull requests to the Git repositories are not accepted. Translations are synced manually between Weblate and the Godot repositories by maintainers.

You should therefore register on Weblate [https://hosted.weblate.org/accounts/register/] to contribute to Godot's translations.

Once signed in, browse to the Godot resource which you want to contribute to (in this page we will use the editor translation [https://hosted.weblate.org/projects/godot-engine/godot/] as an example) to find the list of all languages:


Hosted Weblate
Dashboard
Projects
Languages
Checks
Add
Godot Engine / Godot
translated 46%
Translations
Info
Alerts
Search
Glossaries
Insights
Files
Tools
Manage
Share
Watching

Language	Translated	Untranslated	Untranslated words	Checks	Suggestions	Comments
Afrikaans MIT	7%	2,611	12,048	166	11	
Albanian MIT	21%	2,222	10,327	90	4	
Arabic MIT	91%	229	1,536	127	81	1
Basque MIT	11%	2,520	11,747	5		
Bengali MIT	39%	1,723	8,659	526	3	
Bulgarian MIT	28%	2,038	10,411	76	122	14
Catalan MIT	82%	503	3,324	366	2	2
Chinese (Simplified) MIT	✓			87	1	7
Chinese (Traditional) MIT	✓					

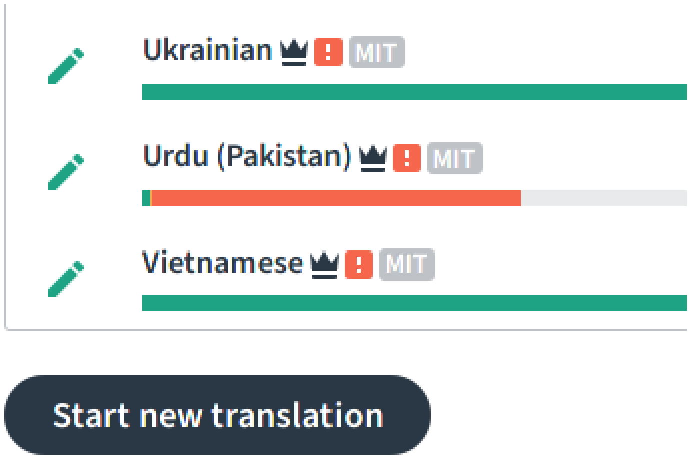
See also

Feel free to consult Weblate's own documentation on the translation workflow [https://docs.weblate.org/en/latest/user/translating.html] for more details.

Adding a new language

If your language is already listed, click on its name to access the overview, and skip the rest of this section.

If your language is not listed, scroll to the bottom of the list of languages and click the "Start new translation" button, and select the language you want to translate to:



Important

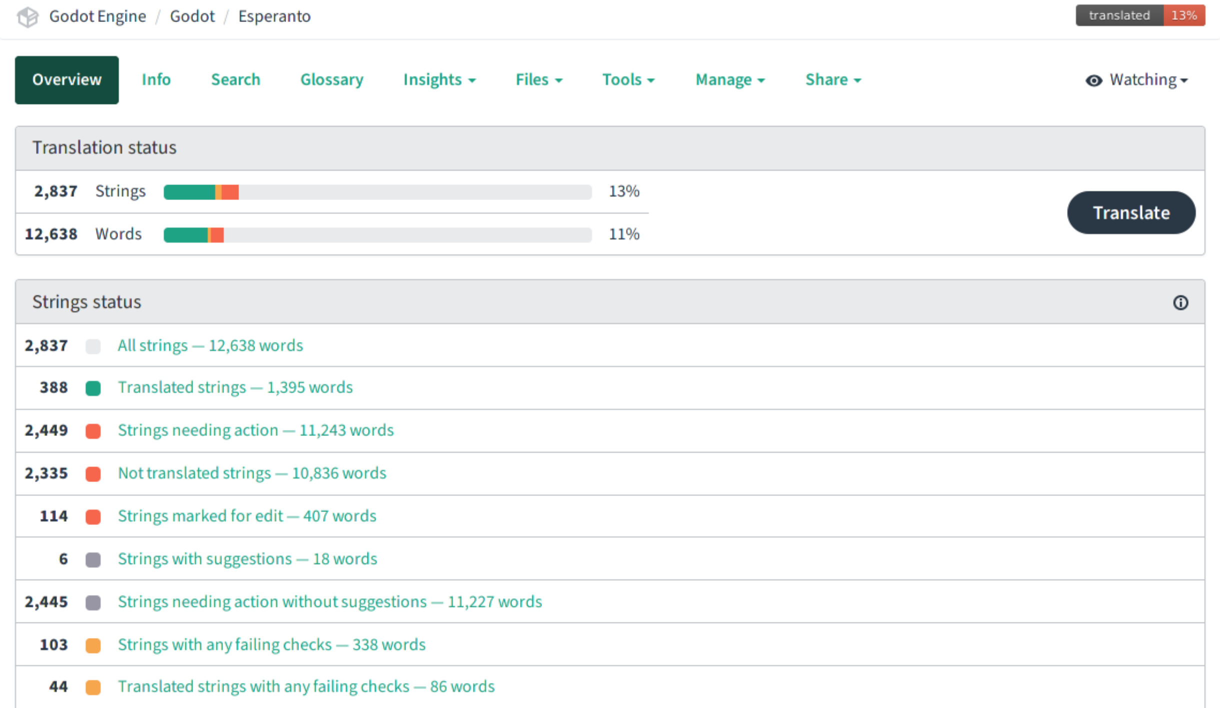
If your language is spoken in several countries with only limited regional variations, please consider adding it with its generic variant (e.g. `fr` for French) instead of a regional variant (e.g. `fr_FR` for French (France), `fr_CA` for French (Canada), or `fr_DZ` for French (Algeria)).

Godot has a huge amount of content to translate, so duplicating the work for regional variants should only be done if the language variations are significant enough. Additionally, if a translation is done with for a regional variant, it will only be available automatically for users located in this region (or having their system language configured for this region).

When regional variations are significant enough to warrant separate translations, we advise to focus on completing a generic variant first if possible, then duplicate the fully completed translation for regional variants and do the relevant edits. This is typically a good strategy for e.g. Spanish (work on `es` first, then duplicate it to `es_AR`, `es_ES`, `es_MX`, etc. if necessary) or Portuguese (`pt_BR` vs `pt_PT`).

Translation interface

Once a language has been selected, you will see an overview of the translation status, including how many strings are left to translate or review. Each item can be clicked and used to browse through the corresponding list. You can also click the "Translate" button to get started on the list of strings needing action.


Godot Engine / Godot / Esperanto
translated 13%
Overview
Info
Search
Glossary
Insights
Files
Tools
Manage
Share
Watching
Translate

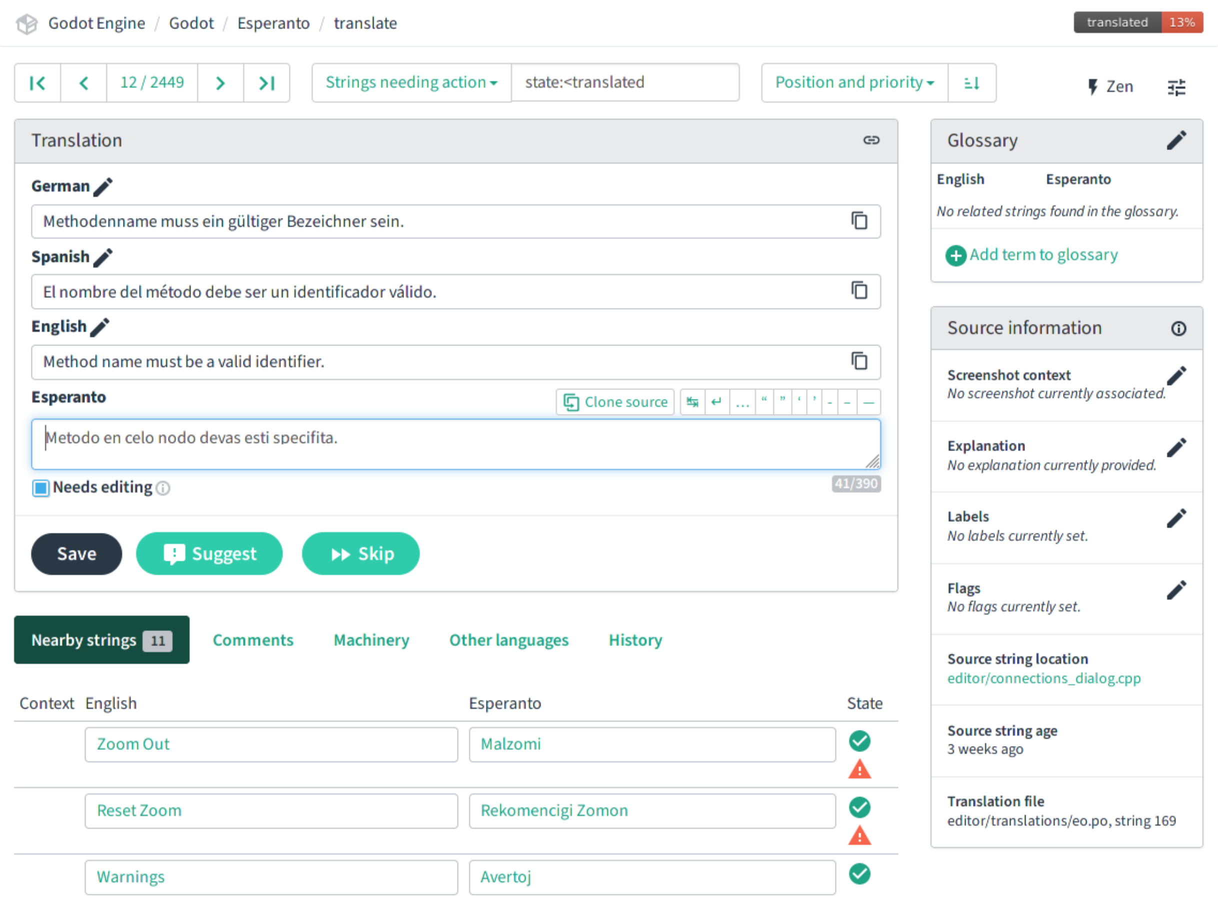
Translation status

2,837	Strings	13%
12,638	Words	11%

Strings status

2,837	⬜	All strings — 12,638 words
388	🟩	Translated strings — 1,395 words
2,449	🟥	Strings needing action — 11,243 words
2,335	🟥	Not translated strings — 10,836 words
114	🟥	Strings marked for edit — 407 words
6	▨	Strings with suggestions — 18 words
2,445	▨	Strings needing action without suggestions — 11,227 words
103	🟧	Strings with any failing checks — 338 words
44	🟧	Translated strings with any failing checks — 86 words

After selecting a list of clicking "Translate", you will see the main translation interface where all the work happens:


Godot Engine / Godot / Esperanto / translate
translated 13%
12 / 2449
Strings needing action
Position and priority
Zen
Translation
Glossary
English
Esperanto
Add term to glossary
Source information
Clone source
Save
Suggest
Skip
Nearby strings 11
Comments
Machinery
Other languages
History
Context
English
Esperanto
State
41/390

state:<translated

German

Methodenname muss ein gültiger Bezeichner sein.

Spanish

El nombre del método debe ser un identificador válido.

English

Method name must be a valid identifier.

Esperanto

Metodo en celo nodo devas esti specifita.

☑ Needs editing

No related strings found in the glossary.

Screenshot context
No screenshot currently associated.

Explanation
No explanation currently provided.

Labels
No labels currently set.

Flags
No flags currently set.

Source string location
editor/connections_dialog.cpp

Source string age
3 weeks ago

Translation file
editor/translations/eo.po, string 169

English	Esperanto
Zoom Out	Malzomi
Reset Zoom	Rekomencigi Zomon
Warnings	Avertoj

On that page, you have:

- A toolbar which lets you cycle through strings of the current list, change to another predefined list or do a custom search, etc. There is also a "Zen" editing mode with a simplified interface.
- The actual string you are working on in the "Translation" panel. By default, there should be the English source string and an edit box for your language. If you are familiar with other languages, you can add them in your user settings to give you more context for translation. Once you are done editing the current string, press "Save" to confirm changes and move to the next entry. Alternatively, use the "Skip" button to skip it. The "Needs editing" checkbox means that the original string was updated, and the translation therefore needs review to take those changes into account

(in PO jargon, these are so-called "fuzzy" strings). Such strings won't be used in the translation until fixed.
- The bottom panel has various tools which can help with the translation effort, such as context from nearby strings (usually from the same editor tool or documentation page, so they might use similar terms), comments from other translators, machine translations, and a list of all other existing translations for that string.
- On the top right, the glossary shows terms for which an entry has been added previously, and which are included in the current string. For example, if you decided with fellow translators to use a specific translation for the "node" term in Godot, you can add it to the glossary to ensure that other translators use the same convention.
- The bottom right panel includes information on the source string. The most relevant item is the "source string location", which links you to the original string on GitHub. You may need to search for the string in the page to locate it and its surrounding context.

Locating original content

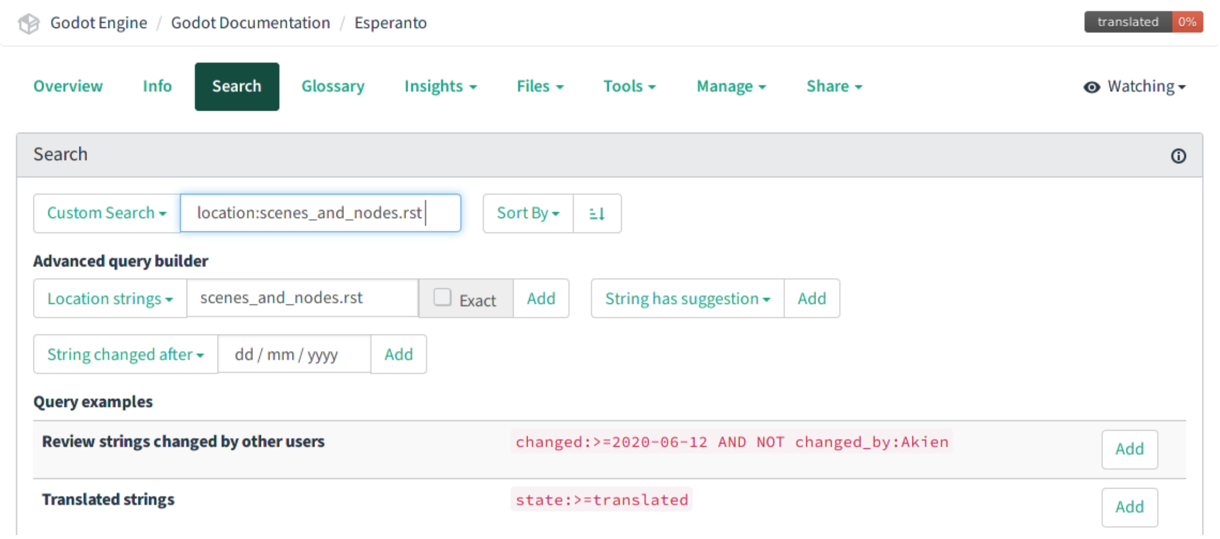
PO files are an ordered list of source strings (`msgid`) and their translation (`msgstr`), and by default, Weblate will present the strings in that order. It can therefore be useful to understand how the content is organized in the PO files to help you locate the original content and use it as a reference when translating.

Important

It is primordial to use the original context as reference when translating, as many words have several possible translations depending on the context. Using the wrong translation can actually be detrimental to the user and make things harder to understand than if they stayed in English. Using the context also makes the translation effort much easier and more enjoyable, as you can see directly if the translation you wrote will make sense in context.

- The editor interface's translation template is generated by parsing all the C++ source code in **alphabetical order**, so all the strings defined in a given file will be grouped together. For example, if the "source string location" indicates `editor/code_editor.cpp`, the current string (and the nearby ones) is defined in the `editor/code_editor.cpp` code file, and is thereby related to the code editors in Godot (GDScript, shaders).
- The online documentation's translation template is generated from the source RST files in the same order as seen in the **table of contents**, so for example the first strings are from the front page of the documentation. The recommended workflow is therefore to find a unique string corresponding to a page that you want to translate, and then translate all the strings with the same source string location while comparing with the online version of that page in English. An example of source string location could be `getting_started/step_by_step/nodes_and_scenes.rst` for the page Nodes and Scenes.
- The class reference's translation template is generated from the source XML files in **alphabetical order**, which is also the same as the order of the table of contents for the online version. You can therefore locate the source string corresponding to the brief description of a given class to find the first string to translate and all other descriptions from that class should be in the subsequent strings on Weblate. For example, the descriptions for the Node2D class would have the source string location `doc/classes/Node2D.xml`.

A handy tool to locate specific pages/classes is to use Weblate's advanced search feature, and especially the "Location strings" query (which can also be used with the `location:` token, e.g. `location:nodes_and_scenes.rst`):

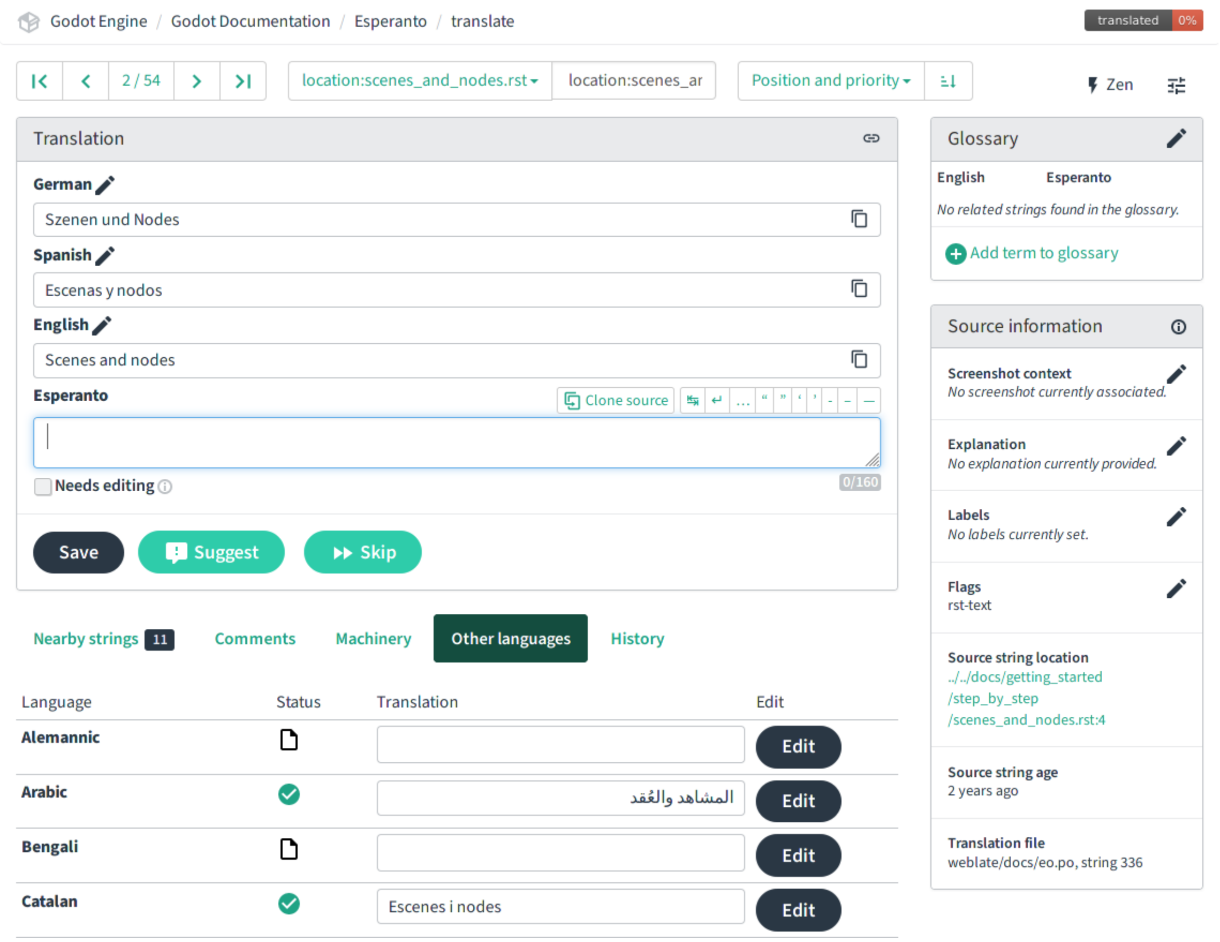

Godot Engine / Godot Documentation / Esperanto
translated 0%
Overview
Info
Search
Glossary
Insights
Files
Tools
Manage
Share
Watching
Search
Custom Search
Sort By
Advanced query builder
Location strings
☐ Exact
Add
String has suggestion
Add
String changed after
Add
Query examples

Search: location:scenes_and_nodes.rst

Location strings: scenes_and_nodes.rst

String changed after: dd / mm / yyyy

Query examples		
Review strings changed by other users	`changed:>=2020-06-12 AND NOT changed_by:Akien`	Add
Translated strings	`state:>=translated`	Add


Godot Engine / Godot Documentation / Esperanto / translate
translated 0%
2 / 54
location:scenes_and_nodes.rst
location:scenes_ar
Position and priority
Zen
Translation
Glossary
English
Esperanto
Add term to glossary
Clone source
Save
Suggest
Skip
Nearby strings 11
Comments
Machinery
Other languages
History
Source information

German: Szenen und Nodes

Spanish: Escenas y nodos

English: Scenes and nodes

Esperanto:

0/160

☐ Needs editing

No related strings found in the glossary.

Language	Status	Translation	Edit
Alemannic			Edit
Arabic	✓	المشاهد والعُقد	Edit
Bengali			Edit
Catalan	✓	Escenes i nodes	Edit

Screenshot context
No screenshot currently associated.

Explanation
No explanation currently provided.

Labels
No labels currently set.

Flags
rst-text

Source string location
../../docs/getting_started/step_by_step/scenes_and_nodes.rst:4

Source string age
2 years ago

Translation file
weblate/docs/eo.po, string 336

Note

When a given source string is used in multiple source locations, they will all be concatenated into one. For example, the above `location:nodes_and_scenes.rst` query would land first on the "Introduction" source string which is used in dozens of pages, including some that come before `nodes_and_scenes.rst` in the template. Clicking the "Next" button then brings us to the "Scene and nodes" title string displayed above. So it may happen that a given paragraph or section title is not at the location you'd expect it when reading the online version of a page.

Respecting the markup syntax

Each translation resource originates from a different source code format, and having some notions on the markup language used for each resource is important to avoid creating syntax errors in your translations.

Editor interface (C++)

The editor translations originate from C++ strings, and may use:

- **C format specifiers** such as `%s` (a string) or `%d` (a number). These specifiers are replaced by content at runtime, and should be preserved and placed in your translation where necessary for it to be meaningful after substitution. You may need to refer to the source string location to understand what kind of content will be substituted if it's not clear from the sentence. Example (`%s` will be substituted with a file name or path):

  ```
  # PO file:
  "There is no '%s' file."
  ```

```
# Weblate:
There is no '%s' file.
```

- **C escape characters** such as `\n` (line break) or `\t` (tabulation). In the Weblate editor, the `\n` characters are replaced by ↵ (return) and `\t` by ↹. Tabs are not used much, but you should make sure to use line breaks in the same way as the original English string (Weblate will issue a warning if you don't). Line breaks might sometimes be used for vertical spacing, or manual wrapping of long lines which would otherwise be too long especially in the editor translation). Example:

```
# PO file:
"Scene '%s' is currently being edited.\n"
"Changes will only take effect when reloaded."

# Weblate:
Scene '%s' is currently being edited.↵
Changes will only take effect when reloaded.
```

Note

Only logical order of the characters matters, in the right-to-left text, format specifiers may be displayed as `s%`.

Online documentation (RST)

The documentation translations originate from reStructuredText (RST) files, which also use their own markup syntax to style text, create internal and external links, etc. Here are some examples:

```
# "development" is styled bold.
# "Have a look here" is a link pointing to
https://docs.godotengine.org/en/latest.
# You should translate "Have a look here", but not the URL,
unless there is
```

```
# a matching URL for the same content in your language.
# Note: The `, <, >, and _ characters all have a meaning in the
hyperlink
# syntax and should be preserved.

Looking for the documentation of the current **development**
branch?
`Have a look here <https://docs.godotengine.org/en/latest>`_.

# "|supported|" is an inline reference to an image and should
stay unchanged.
# "master" uses the markup for inline code, and will be styled
as such.
# Note: Inline code in RST uses 2 backticks on each side,
unlike Markdown.
# Single backticks are used for hyperlinks.

|supported| Backwards-compatible new features (backported from
the ``master``
branch) as well as bug, security, and platform support fixes.

# The :ref: Sphinx "role" is used for internal references to
other pages of
# the documentation.
# It can be used with only the reference name of a page (which
should not be
# changed), in which case the title of that page will be
displayed:

See :ref:`doc_ways_to_contribute`.

# Or it can be used with an optional custom title, which should
thus be translated:

See :ref:`how to contribute <doc_ways_to_contribute>`.

# You may encounter other Sphinx roles, such as :kbd: used for
shortcut keys.
# You can translate the content between backticks to match the
usual key names,
# if it's different from the English one.

Save the scene. Click Scene -> Save, or press :kbd:`Ctrl + S`
```

```
on Windows/Linux
or :kbd:`Cmd + S` on macOS.
```

See also

See Sphinx's reStructured Text primer [https://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html] for a quick overview of the markup language you may find in source strings. You may encounter especially the inline markup (bold, italics, inline code) and the internal and external hyperlink markup.

Class reference (BBCode)

The class reference is documented in the main Godot repository using XML files, and with BBCode-like markup for styling and internal references.

Some of the tags used are from the original BBCode (e.g. `[b]Bold[/b]` and `[i]Italics[/i]`), while others are Godot-specific and used for advanced features such as inline code (e.g. `[code]true[/code]`), linking to another class (e.g. `[Node2D]`) or to a property in a given class (e.g. `[member Node2D.position]`), or for multiline code blocks. Example:

```
Returns a color according to the standardized [code]name[/code]
with [code]alpha[/code] ranging from 0 to 1.
[codeblock]
red = ColorN("red", 1)
[/codeblock]
Supported color names are the same as the constants defined in
[Color].
```

In the above example, `[code]name[/code]`, `[code]alpha[/code]`, and `[Color]` should *not* be translated, as they refer respectively to argument names and a class of the Godot API. Similarly, the contents of the `[codeblock]` should not be translated, as `ColorN` is a function of the Godot API and `"red"` is one of the named colors it supports. At most, you can translate the name of the variable which holds the result (`red = ...`).

Note also that in the XML, each line is a paragraph, so you should not add line breaks if they are not part of the original translation.

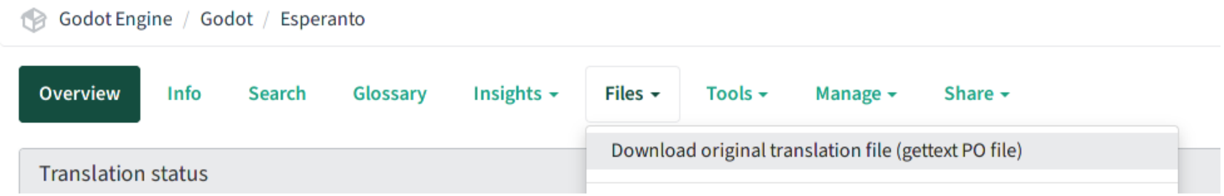
See also

See our documentation for class reference writers for the list of BBCode-like tags which are used throughout the class reference.

Offline translation and testing

While we advise using the Weblate interface to write translations, you also have the possibility to download the PO file locally to translate it with your preferred PO editing application, such as Poedit [https://poedit.net/] or Lokalize [https://userbase.kde.org/Lokalize].

To download the PO file locally, browse to the translation overview for your language, and select the first item in the "Files" menu:


Godot Engine / Godot / Esperanto
Overview
Info
Search
Glossary
Insights
Files
Tools
Manage
Share
Download original translation file (gettext PO file)
Translation status

Once you are done with a series of edits, use the "Upload translation" item in that same menu and select your file. Choose "Add as translation" for the file upload mode.

Note

If a significant amount of time has passed between your download of the PO file and the upload of the edited version, there is a risk to overwrite the translations authored by other contributors in the meantime. This is why we advise to use the online interface so that you always work on the latest version.

If you want to test changes locally (especially for the editor translation), you can use the downloaded PO file and compile Godot from source.

Rename the editor translation PO file to `<lang>.po` (e.g. `eo.po` for Esperanto) and place it in the `editor/translations/` folder (GitHub [https://github.com/godotengine/godot/tree/master/editor/translations]).

You can also test class reference changes the same way by renaming the PO file similarly and placing it in the `doc/translations/` folder (GitHub [https://github.com/godotengine/godot/tree/master/doc/translations]).

Localizing documentation images

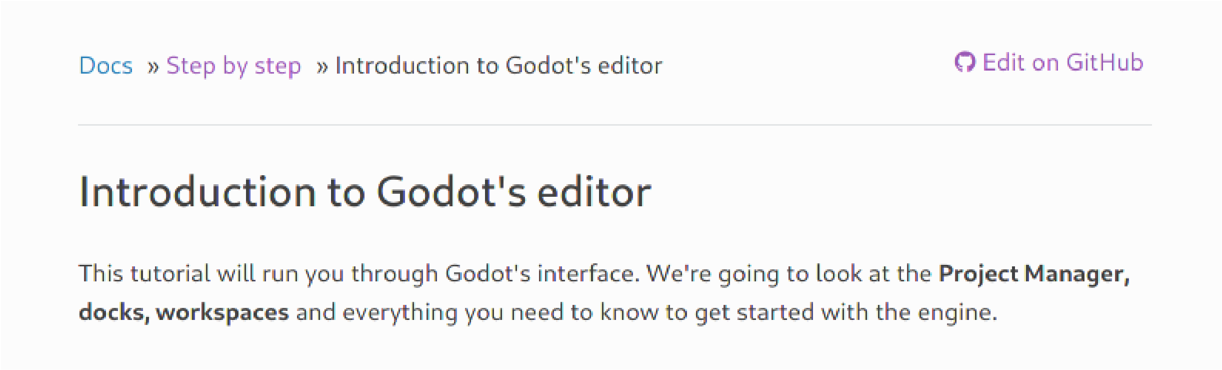
The online documentation includes many images, which can be screenshots of the Godot editor, custom-made graphs, of any other kind of visual content. Some of it includes text and might thus be relevant to localize in your language.

This part is not handled via Weblate, but directly on the godot-docs-l10n [https://github.com/godotengine/godot-docs-l10n] Git repository where the documentation translations are synced from Weblate.

Note

The workflow is not the most straightforward and requires some knowledge of Git. We plan to work on a simplified Web tool which could be used to manage image localization in a convenient way, abstracting away these steps.

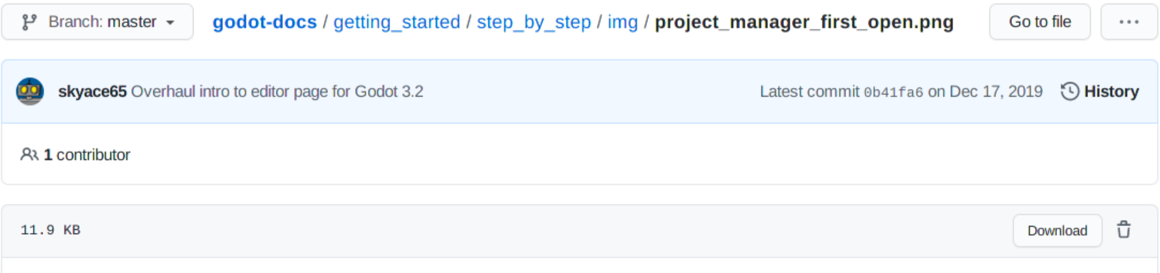
To translate an image, you should first locate it in the original English documentation. To do so, browse the relevant page in the docs, e.g. First look at Godot's interface. Click the "Edit on GitHub" link in the top right corner:


Docs » Step by step » Introduction to Godot's editor
Edit on GitHub

Introduction to Godot's editor

This tutorial will run you through Godot's interface. We're going to look at the **Project Manager, docks, workspaces** and everything you need to know to get started with the engine.

On GitHub, click on the image you want to translate. If relevant, click on "Download" to download it locally and edit it with an image edition tool. Note the full path to the image as it will be needed further down (here `getting_started/step_by_step/img/project_manager_first_open.png`).


Branch: master
godot-docs / getting_started / step_by_step / img / project_manager_first_open.png
Go to file
···
History
Download

skyace65 Overhaul intro to editor page for Godot 3.2

Latest commit 0b41fa6 on Dec 17, 2019

1 contributor

11.9 KB

Create your localized version of the image, either by editing the English one, or by taking a screenshot of the editor with your language, if it's an editor screenshot. Some images may also have source files available in SVG format, so you can browse the `img/` folder which contains them to check for that.

Name your localized image like the original one, but with the language code added before the extension, e.g. `project_manager_first_open.png` would become `project_manager_first_open.fr.png` for the French localization.

Finally, on godot-docs-l10n [https://github.com/godotengine/godot-docs-l10n], recreate the same folder structure as for the original image in the `images` subfolder (GitHub [https://github.com/godotengine/godot-docs-l10n/tree/master/images]), and place your translated image there. In our example, the end result should be `images/getting_started/step_by_step/img/project_manager_first_open.fr.png`.

Repeat this for other images and make a Pull Request.

Asset Library

- About the Asset Library
- Using the Asset Library
- Submitting to the Asset Library

About the Asset Library

The Godot Asset Library, otherwise known as the AssetLib, is a repository of user-submitted Godot addons, scripts, tools, and other resources, collectively referred to as assets. They're available to all Godot users for download directly from within the engine, but it can also be accessed at Godot's official website [https://godotengine.org/asset-library/asset].

On the surface, the Asset Library might look and function similar to asset stores available for other engines, such as Unity's Asset Store, or Unreal Engine's Marketplace, where you can submit both freely-available assets, as well as paid, commercial ones. In addition, often times such assets are distributed under non-free, proprietary licenses, limiting what you can do with them.

The Asset Library is different - all assets are distributed free of charge, and under a host of open source licenses (such as the MIT license, the GPL, and the Boost Software License). This makes the AssetLib more similar to the software repositories of a Linux distribution.

This set of pages will cover how to use the AssetLib (both from inside Godot, and on the website), how you can submit your own assets, and what the guidelines for submission are.

Please note that the AssetLib is relatively young - it may have various pain points, bugs, and usability issues. As with all Godot projects, the code repository is available on GitHub [https://github.com/godotengine/godot-asset-library], where you can submit pull requests and issues, so please do not hesitate to visit it!

Types of assets

Be aware that there are, broadly, two different types of assets you can post.

- Assets labeled as "Templates", "Projects", or "Demos" appear under the "Asset Library" tab in the Godot Project Manager. These assets are standalone Godot projects that can run by themselves.
- Other assets show up inside of the Godot editor under the "Asset Library" main screen tab, next to "2D", "3D", and "Script". These assets are meant to be downloaded and placed into an existing Godot project.

Frequently asked questions

Can paid assets be uploaded to the asset library?

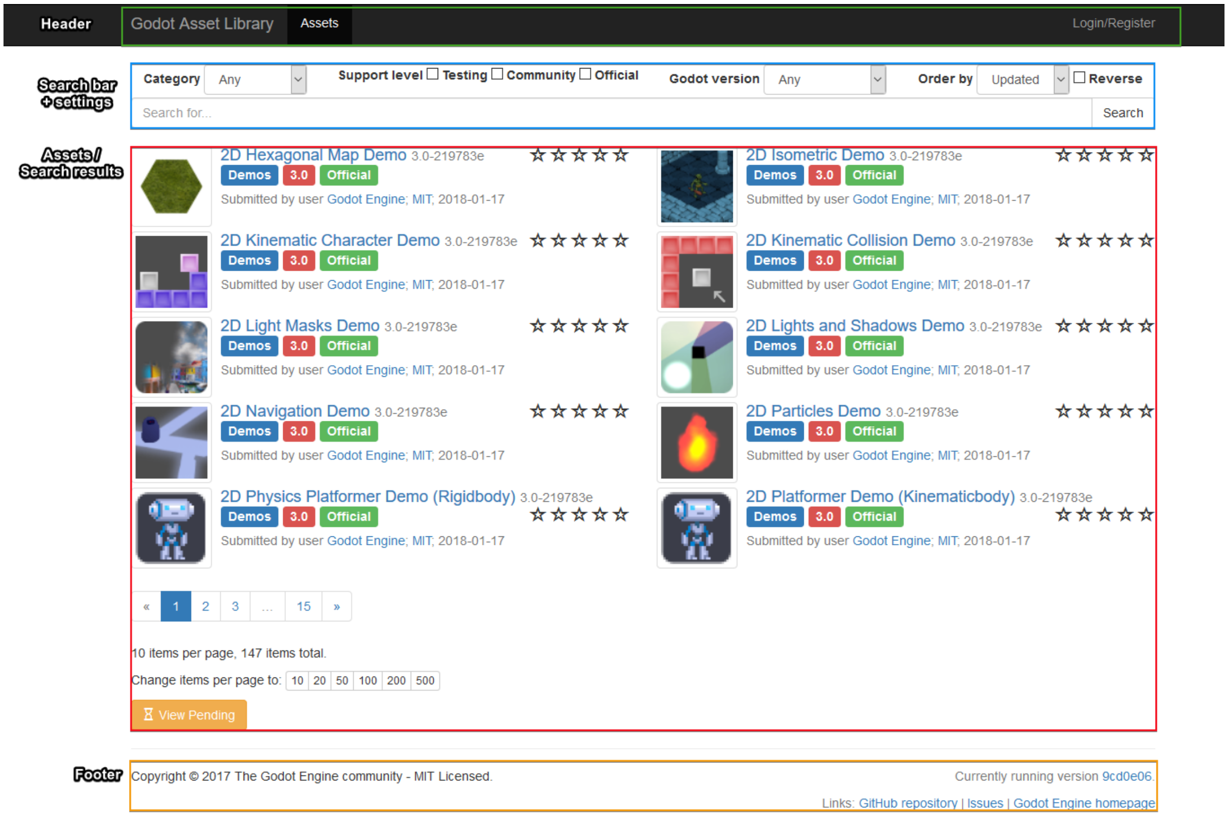
Not to the official one, though in the future, there might be other asset libraries which allow it. That said, you are allowed to monetize and sell Godot assets outside the Asset Library.

Using the Asset Library

On the website

Overview

As mentioned before, you can access the web frontend of the Asset Library on [Godot's official website](https://godotengine.org/asset-library/asset) [https://godotengine.org/asset-library/asset]. This is what it looks like when you first visit it:



Category: Any
Support level: ☐ Testing ☐ Community ☐ Official
Godot version: Any
Order by: Updated ☐ Reverse
Search for...
Search

2D Hexagonal Map Demo 3.0-219783e
Demos 3.0 Official
Submitted by user Godot Engine; MIT; 2018-01-17

2D Isometric Demo 3.0-219783e
Demos 3.0 Official
Submitted by user Godot Engine; MIT; 2018-01-17

2D Kinematic Character Demo 3.0-219783e
Demos 3.0 Official
Submitted by user Godot Engine; MIT; 2018-01-17

2D Kinematic Collision Demo 3.0-219783e
Demos 3.0 Official
Submitted by user Godot Engine; MIT; 2018-01-17

2D Light Masks Demo 3.0-219783e
Demos 3.0 Official
Submitted by user Godot Engine; MIT; 2018-01-17

2D Lights and Shadows Demo 3.0-219783e
Demos 3.0 Official
Submitted by user Godot Engine; MIT; 2018-01-17

2D Navigation Demo 3.0-219783e
Demos 3.0 Official
Submitted by user Godot Engine; MIT; 2018-01-17

2D Particles Demo 3.0-219783e
Demos 3.0 Official
Submitted by user Godot Engine; MIT; 2018-01-17

2D Physics Platformer Demo (Rigidbody) 3.0-219783e
Demos 3.0 Official
Submitted by user Godot Engine; MIT; 2018-01-17

2D Platformer Demo (Kinematicbody) 3.0-219783e
Demos 3.0 Official
Submitted by user Godot Engine; MIT; 2018-01-17

« 1 2 3 … 15 »

10 items per page, 147 items total.

Change items per page to: 10 20 50 100 200 500

View Pending

Copyright © 2017 The Godot Engine community - MIT Licensed.

Currently running version 9cd0e06.

Links: GitHub repository | Issues | Godot Engine homepage

At the top, you see the **header**, which takes you to various other parts of the AssetLib - at the moment, it's empty, as we are not logged in.

Searching

In the center is the **search bar + settings** section, and the **assets** section below it - this lets you filter out certain kinds of assets based on a host of criteria. These include the asset **category** (such as 2D tools, scripts and demos), **engine version** they are intended for, **sorting order** (by update date, by name, etc.) and **support level**.

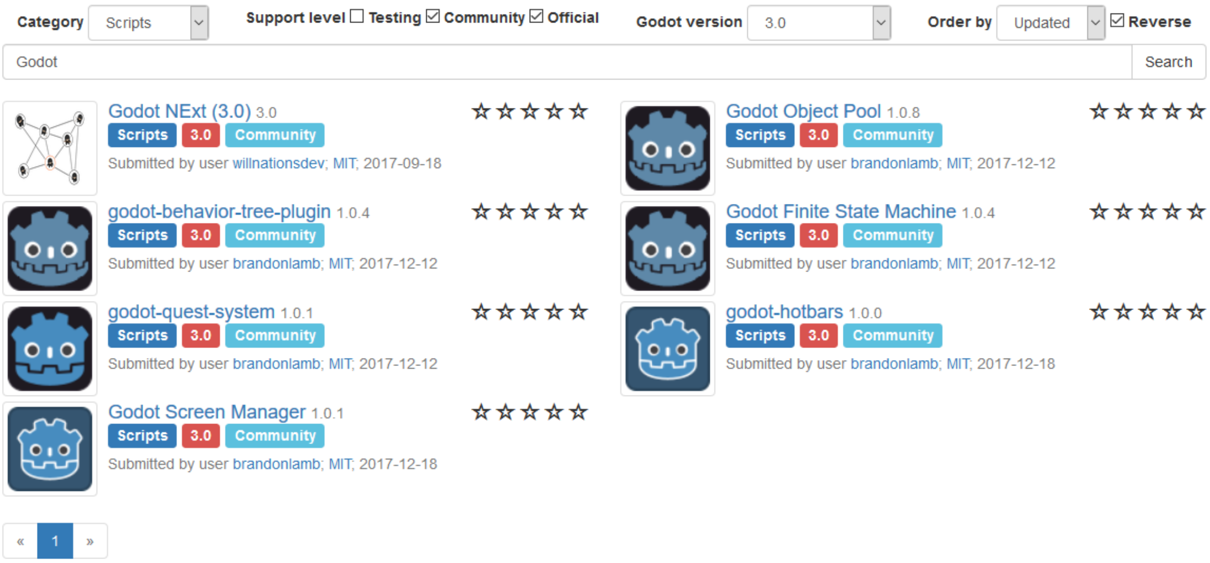
While most other filter settings should be fairly self-explanatory, it's worth going over what "support level" means in the Asset Library. Currently there are three support levels, and each asset can belong to only one.

Official assets are created and maintained by the official Godot Engine developers. Currently, these include the official engine demos, which showcase how various areas of the engine work.

Community assets are submitted and maintained by the members of the Godot community.

Testing assets are works-in-progress, and may contain bugs and usability issues. They are not recommended for use in serious projects, but you are encouraged to download, test them, and submit issues to the original authors.

You can mix and match any of the search filters and criteria, and upon clicking the Search button, receive the list of all assets in the Library that match them.



Category Scripts

Support level ☐ Testing ☑ Community ☑ Official

Godot version 3.0

Order by Updated ☑ Reverse

Godot Search

Godot NExt (3.0) 3.0 Scripts 3.0 Community ☆☆☆☆☆
Submitted by user willnationsdev; MIT; 2017-09-18

Godot Object Pool 1.0.8 Scripts 3.0 Community ☆☆☆☆☆
Submitted by user brandonlamb; MIT; 2017-12-12

godot-behavior-tree-plugin 1.0.4 Scripts 3.0 Community ☆☆☆☆☆
Submitted by user brandonlamb; MIT; 2017-12-12

Godot Finite State Machine 1.0.4 Scripts 3.0 Community ☆☆☆☆☆
Submitted by user brandonlamb; MIT; 2017-12-12

godot-quest-system 1.0.1 Scripts 3.0 Community ☆☆☆☆☆
Submitted by user brandonlamb; MIT; 2017-12-12

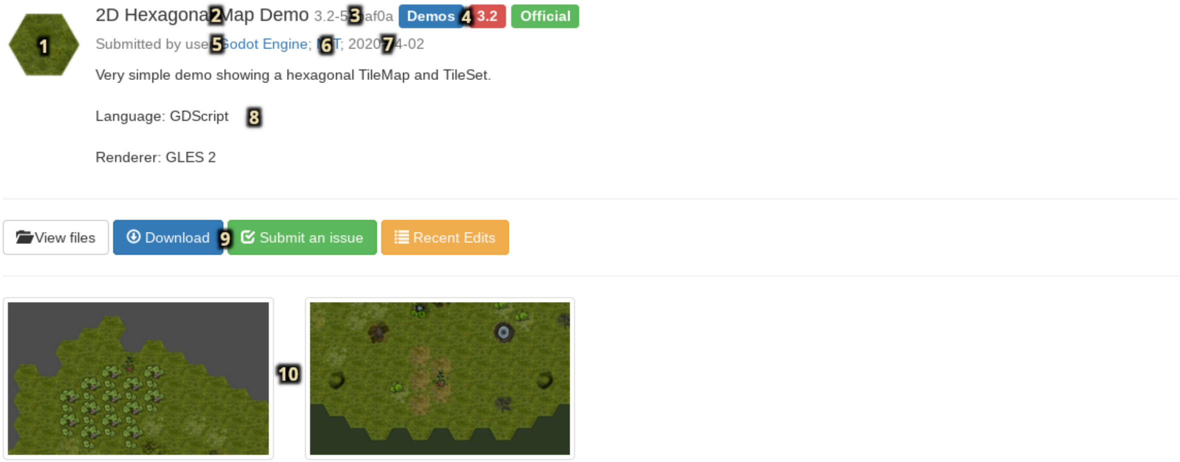
godot-hotbars 1.0.0 Scripts 3.0 Community ☆☆☆☆☆
Submitted by user brandonlamb; MIT; 2017-12-18

Godot Screen Manager 1.0.1 Scripts 3.0 Community ☆☆☆☆☆
Submitted by user brandonlamb; MIT; 2017-12-18

Note that the search results are not updated in real-time, so you will have to re-submit the search query each time you change the query settings.

Breakdown of an asset

Now let's take a look at what an asset's page looks like and what it contains.



2D Hexagona Map Demo 3.2-5 af0a Demos 3.2 Official
Submitted by use odot Engine; T; 2020 4-02
Very simple demo showing a hexagonal TileMap and TileSet.

Language: GDScript

Renderer: GLES 2

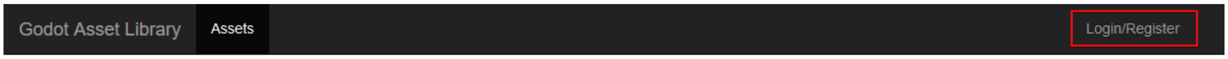
1. Asset's thumbnail/icon.
2. Asset's name.

3. Current version number of the asset.
4. Asset's category, Godot version, and support status.
5. Asset's original author/submitter.
6. The license the asset is distributed under.
7. The date of the asset's latest edit/update.
8. A textual description of the asset.
9. Links related to the asset (download link, file list, issue tracker).
10. Images and videos showcasing the asset.

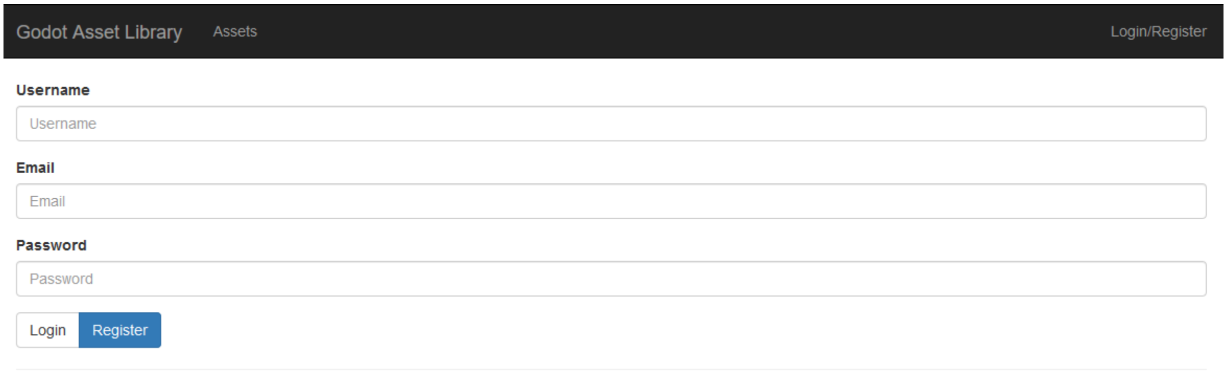
Registering and logging in

In order to upload assets to the AssetLib, you need to be logged in, and to do that, you need a registered user account. In the future, this may also give you access to other features, such as commenting on or rating the existing assets. You do *not* need to be logged in to browse and download the assets.

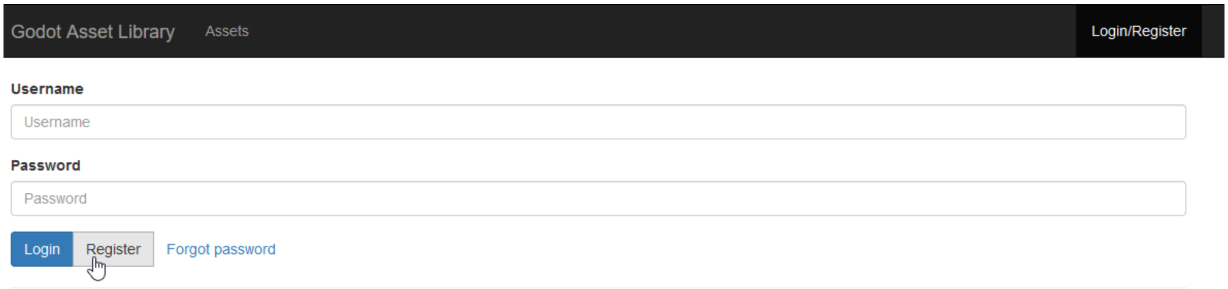
The login/registration page can be accessed from the AssetLib header.


Godot Asset Library
Assets
Login/Register

From here, you can register your account, which requires a valid email address, a username, and a (preferably strong) password.

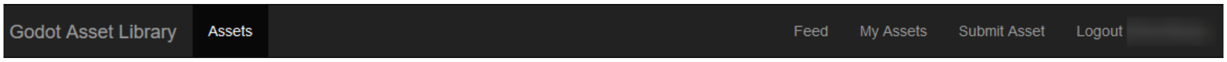

Godot Asset Library
Assets
Login/Register
Username
Username
Email
Email
Password
Password
Login
Register

Then, you can use your username and password to log in.


Godot Asset Library
Assets
Login/Register
Username
Username
Password
Password
Login
Register
Forgot password

This will change the look of the AssetLib header. Now you get access to a handful of new functions:

- The feed, which shows a list of status updates on your submitted assets (and possibly more in the future).
- A list of your uploaded assets.
- The ability to submit new assets.


Godot Asset Library
Assets
Feed
My Assets
Submit Asset
Logout
[redacted]

You can learn how to submit assets to the Library, and what the asset submission guidelines are, in the next part of this tutorial, Submitting to the Asset Library.

In the editor

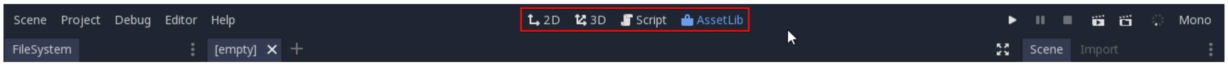
Note

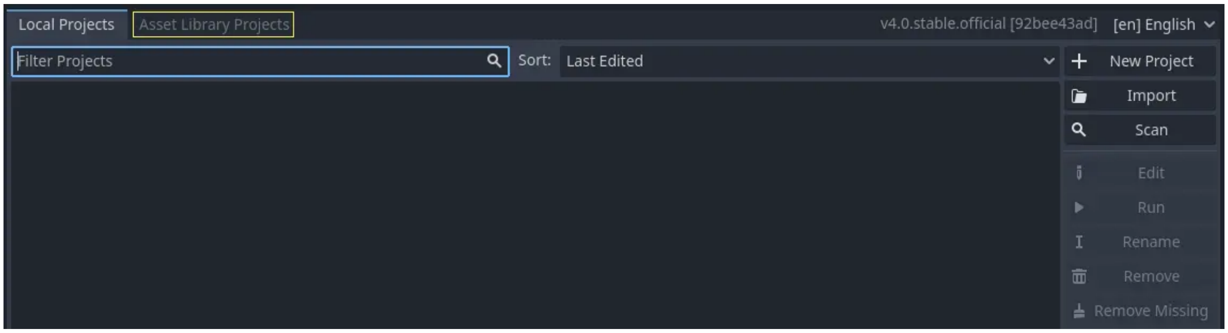
The editor will display different categories of assets depending on whether you're browsing the Project Manager's **Asset Library Projects** tab or the editor's **AssetLib** tab.

The Project Manager's **Asset Library Projects** tab will only display assets that are standalone projects by themselves. This is denoted on the asset library with the *Templates*, *Demos* and *Projects* categories.

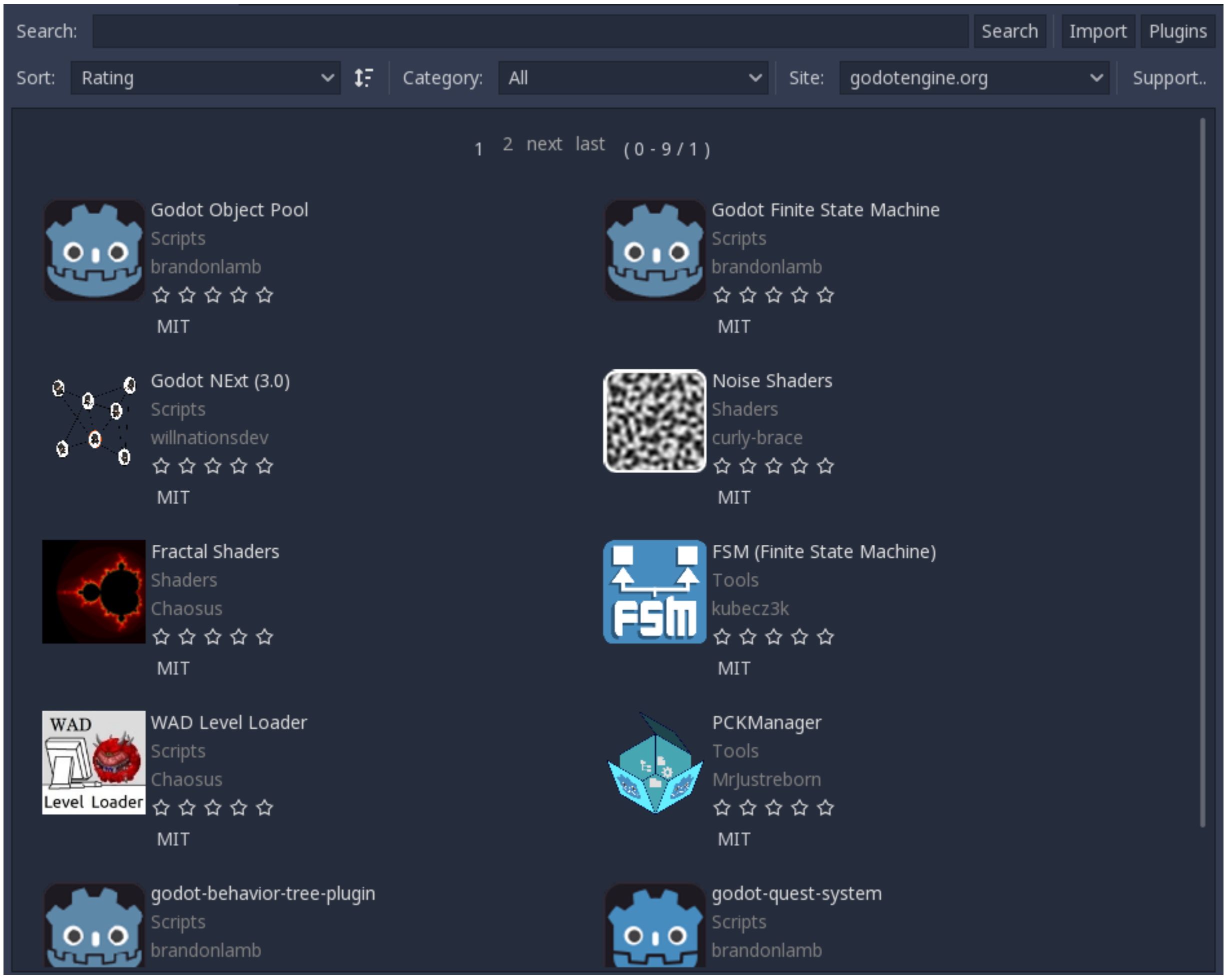
The editor's **AssetLib** tab will only display assets that are *not* standalone projects by themselves. In other words, it will display assets from all categories except *Templates*, *Demos* and *Projects*.

You can also access the AssetLib directly from Godot:


Scene
Project
Debug
Editor
Help
2D
3D
Script
AssetLib
Mono
FileSystem
[empty]
Scene
Import


Local Projects
Asset Library Projects
v4.0.stable.official [92bee43ad]
[en] English
Filter Projects
Sort:
Last Edited
New Project
Import
Scan
Edit
Run
Rename
Remove
Remove Missing

Click on it, and Godot will fetch info about the assets from the AssetLib. Once it's finished, you will see a window similar to what the AssetLib website looks like, with some differences:

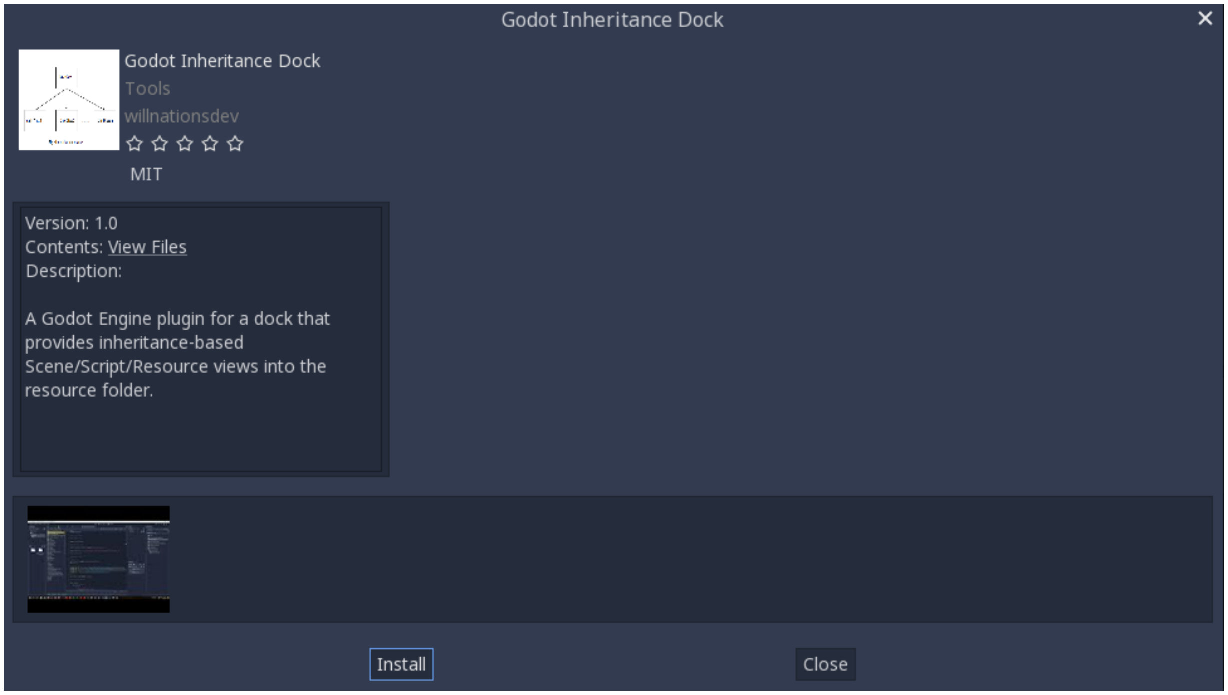

Search:
Search
Import
Plugins
Sort:
Rating
Category:
All
Site:
godotengine.org
Support..
1 2 next last (0 - 9 / 1)
Godot Object Pool
Scripts
brandonlamb
MIT
Godot Finite State Machine
Scripts
brandonlamb
MIT
Godot NExt (3.0)
Scripts
willnationsdev
MIT
Noise Shaders
Shaders
curly-brace
MIT
Fractal Shaders
Shaders
Chaosus
MIT
FSM (Finite State Machine)
Tools
kubecz3k
MIT
WAD Level Loader
Scripts
Chaosus
MIT
PCKManager
Tools
MrJustreborn
MIT
godot-behavior-tree-plugin
Scripts
brandonlamb
godot-quest-system
Scripts
brandonlamb

Similarly to the web version of the AssetLib, here you can search for assets by category, name, and sort them by factors such as name or edit date.

Notably, you can only fetch assets for the current version of Godot you are running. Projects, Demos and Templates can be downloaded from the Project Manager view of the AssetLib. Addons (tools, scripts, materials etc.) can be downloaded from the in-project AssetLib and added to the current project. In addition, unlike when using the web frontend, the search results are updated in real-time (you do not have to press Search after every change to your search query for the changes to take place).

In the future, you will be able to choose a different AssetLib provider to fetch assets from (using the Site dropdown menu), however currently only the official Godot website [https://godotengine.org] version of the AssetLib is supported, as well as the version that may be running on your local machine's web server (the localhost option).

When you click on an asset, you will see more information about it.

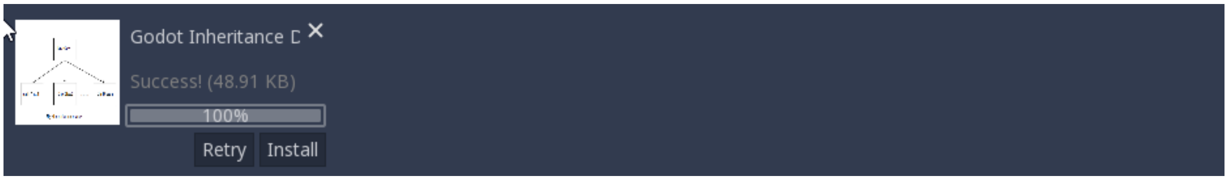

Godot Inheritance Dock
Godot Inheritance Dock
Tools
willnationsdev
☆ ☆ ☆ ☆ ☆
MIT

Version: 1.0
Contents: View Files
Description:

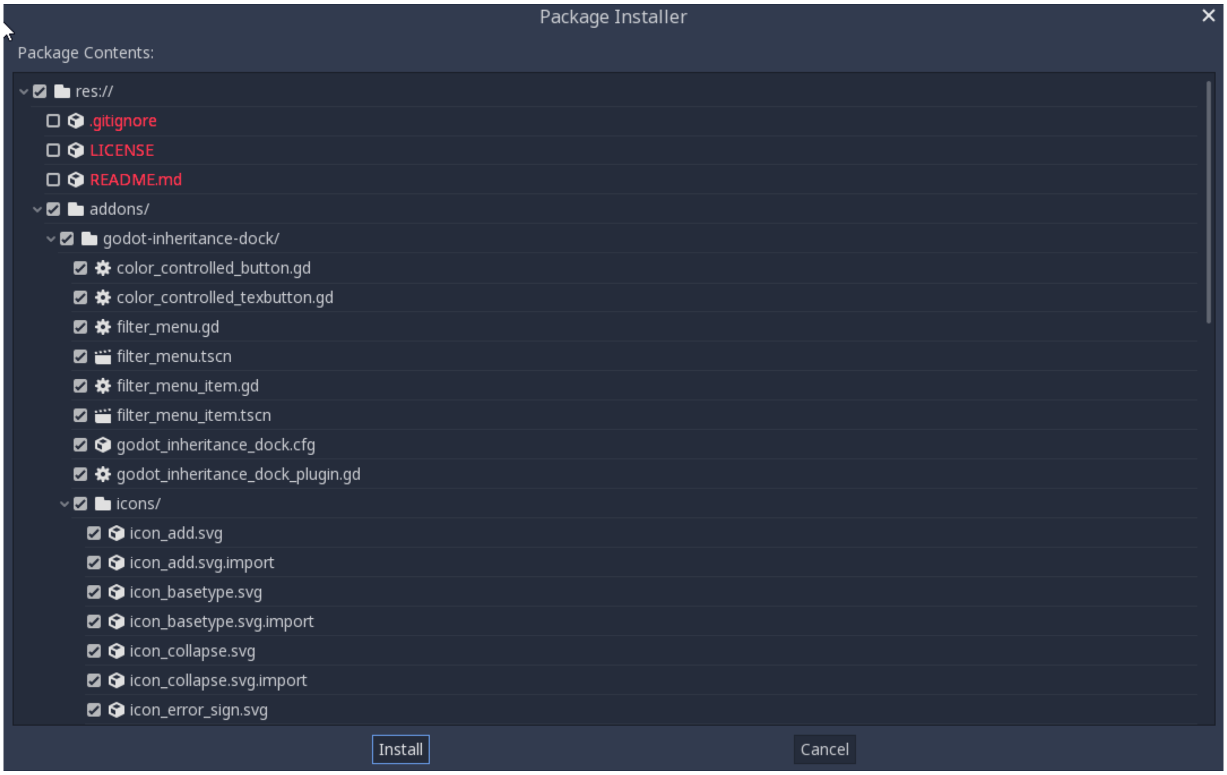
A Godot Engine plugin for a dock that provides inheritance-based Scene/Script/Resource views into the resource folder.

Install
Close

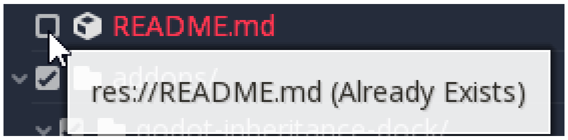
If you click on the Install button, Godot will fetch an archive of the asset, and will track download progress of it at the bottom of the editor window. If the download fails, you can retry it using the Retry button.


Godot Inheritance D
Success! (48.91 KB)
100%
Retry
Install

When it finishes, you can proceed to install it using the Install button. This will bring up the Package Installer window.

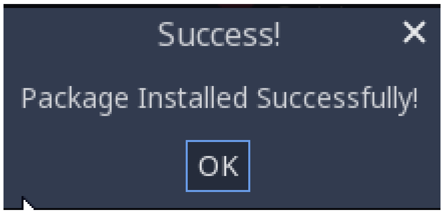

Package Installer
Package Contents:
☑ res://
☐ .gitignore
☐ LICENSE
☐ README.md
☑ addons/
☑ godot-inheritance-dock/
☑ color_controlled_button.gd
☑ color_controlled_texbutton.gd
☑ filter_menu.gd
☑ filter_menu.tscn
☑ filter_menu_item.gd
☑ filter_menu_item.tscn
☑ godot_inheritance_dock.cfg
☑ godot_inheritance_dock_plugin.gd
☑ icons/
☑ icon_add.svg
☑ icon_add.svg.import
☑ icon_basetype.svg
☑ icon_basetype.svg.import
☑ icon_collapse.svg
☑ icon_collapse.svg.import
☑ icon_error_sign.svg
Install
Cancel

Here you can see a list of all the files that will be installed. You can tick off any of the files that you do not wish to install, and Godot will also inform you about any problems with files that it cannot install. These files will be shown in red, and hovering over them will show you a message stating why it cannot be installed.



res://README.md (Already Exists)

Once you are done, you can press the Install button, which will unzip all the files in the archive, and import any assets contained therein, such as images or 3D models. Once this is done, you should see a message stating that the Package installation is complete.



Package Installed Successfully!

You may also use the Import button to import asset archives obtained elsewhere (such as downloading them directly from the AssetLib web frontend), which will take you through the same package installation procedure as with the assets downloaded directly via Godot that we just covered.

Submitting to the Asset Library

Introduction

This tutorial aims to serve as a guide on how you can submit your own assets to the Godot Asset Library [https://godotengine.org/asset-library/asset] and share them with the Godot community.

As mentioned in the Using the Asset Library document, in order to be able to submit assets to the AssetLib, you need to have a registered account, and be logged in.

Submission guidelines

Before submitting your asset, please ensure it follows all of the requirements, and also consider following the recommendations.

Requirements

Generally speaking, most assets people submit to the asset library are accepted. However, in order for your asset to be accepted, there are a few requirements your asset needs to meet to be approved.

- The asset must **work**. If the asset doesn't run or otherwise doesn't work in the specified Godot version, then it will be rejected.
- The asset must have a proper **.gitignore** file. It's important to keep redundant data out of the repository. Here's a template.

[https://raw.githubusercontent.com/aaronfranke/gitignore/godot/Godot.gitignore]
- No **submodules**, or any submodules must be non-essential. GitHub does not include submodules in the downloaded ZIP file, so if the asset needs the contents of the submodule, your asset won't work.
- The **license** needs to be correct. The license listed on the asset library must match the license in the repository. The repo MUST have a license file, called either "LICENSE" or "LICENSE.md". This file must contain the license text itself and a copyright statement that includes the year(s) and copyright holder.
- Use proper **English** for the name and description of your asset. This includes using correct capitalization, and using full sentences in the description. You can also include other languages, but there should at least be an English version.
- The icon link must be a **direct link**. For icons hosted on GitHub, the link must start with "raw.githubusercontent.com", not "github.com".

Recommendations

These things are not required for your asset to be approved, but if you follow these recommendations, you can help make the asset library a better place for all users.

- When creating non-project assets, it is common practice to place your files inside of an **addons/asset_name/** folder. Do this to avoid having your files clash with other assets, or with the files of users installing your asset. This folder will **not** be automatically generated when a user installs your asset.

- Fix or suppress all script **warnings**. The warning system is there to help identify issues with your code, but people using your asset don't need to see them.

- Make your code conform to the official **style guides**. Having a consistent style helps other people read your code, and it also helps if other people wish to contribute to your asset. See: the GDScript style guide or the C# style guide.

- If you have screenshots in your repo, place them in their own subfolder and add an empty **.gdignore** file in the same folder (note: **gd**, not **git**). This prevents Godot from importing your screenshots. On Windows, open a command prompt in the project folder and run `type nul > .gdignore` to create a file whose name starts with a period.

- If your asset is a library for working with other files, consider including **example files** in the asset.

- Consider adding a **.gitattributes** file to your repo. This file allows giving extra instructions to Git, such as specifying line endings and listing files not required for your asset to function with the `export-ignore` directive. This directive removes such files from the resulting ZIP file, preventing them from being downloaded by the asset library users. These are common examples of **.gitattributes**:

  Projects / Templates

  ```
  # Normalize line endings for all files that Git considers
  text files.
  * text=auto eol=lf
  ```

  Addons / Asset Packs

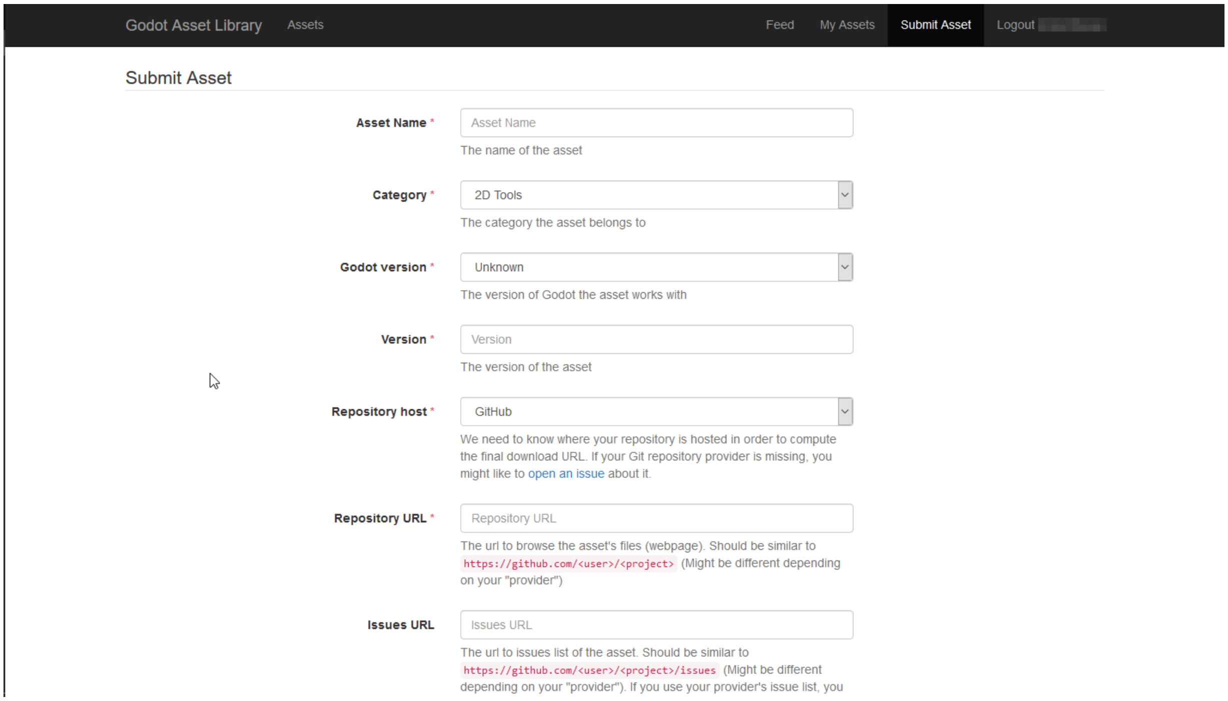
  ```
  # Normalize line endings for all files that Git considers
  text files.
  * text=auto eol=lf

  # Only include the addons folder when downloading from the
  Asset Library.
  /**        export-ignore
  ```

```
/addons     !export-ignore
/addons/** !export-ignore
```

- If you are submitting a plugin, add a **copy** of your license and readme to the plugin folder itself. This is the folder that users are guaranteed to keep with their project, so a copy ensures they always have those files handy (and helps them fulfill your licensing terms).
- While the asset library allows more than just GitHub, consider hosting your asset's source code on **GitHub**. Other services may not work reliably, and a lack of familiarity can be a barrier to contributors.

Submitting

Once you are logged in, you will be able to head over to the "Submit Assets" page of the AssetLib, which will look like this:



Submit Asset

Asset Name * Asset Name
The name of the asset

Category * 2D Tools
The category the asset belongs to

Godot version * Unknown
The version of Godot the asset works with

Version * Version
The version of the asset

Repository host * GitHub
We need to know where your repository is hosted in order to compute the final download URL. If your Git repository provider is missing, you might like to open an issue about it.

Repository URL * Repository URL
The url to browse the asset's files (webpage). Should be similar to `https://github.com/<user>/<project>` (Might be different depending on your "provider")

Issues URL Issues URL
The url to issues list of the asset. Should be similar to `https://github.com/<user>/<project>/issues` (Might be different depending on your "provider"). If you use your provider's issue list, you

While it may look like a lot (and there is more as you scroll down), each field is described in terms of what you should put in. We will nonetheless go over what is required in the submission form here as well.

- **Asset Name**:
  The name of your asset. Should be a unique, descriptive title of what your asset is.

- **Category**:
  The category that your asset belongs to, and will be shown in search results. The category is split into **Addons** and **Projects**. In-editor, assets of the Project type (Templates, Demos, Projects) only show up when viewing the AssetLib from the Project Manager, while assets of the Addon type will only be visible from inside a project.

- **Godot version**:
  The version of the engine that the asset works with. Currently, it's not possible to have a single asset entry contain downloads for multiple engine versions, so you may need to re-submit the asset multiple times, with an entry for each Godot version it supports. This is particularly important when dealing with major versions of the engine, such as Godot 2.x and Godot 3.x.

- **Version**:
  The version number of the asset. While you are free to choose and use any versioning scheme that you like, you may want to look into something such as SemVer [https://semver.org] if you want your asset's versioning scheme to be clear and consistent. Note that there is also an internal version number, incremented every time the asset download URL is changed or updated.

- **Repository host**:

Assets uploaded to the AssetLib are not hosted on it directly. Instead, they point to repositories hosted on third-party Git providers, such as GitHub, GitLab or Bitbucket. This is where you choose which provider your asset uses, so the site can compute the final download link.

- **Repository URL**:

  The URL to your asset's files/webpage. This will vary based on your choice of provider, but it should look similar to *https://github.com/<user>/<project>*.

- **Issues URL**:

  The URL to your asset's issue tracker. Again, this will differ from repository host to repository host, but will likely look similar to *https://github.com/<user>/<project>/issues*. You may leave this field empty if you use your provider's issue tracker, and it's part of the same repository.

- **Download Commit**:

  The commit of the asset. For example, *b1d3172f89b86e52465a74f63a74ac84c491d3e1*. The site computes the actual download URL from this.

- **Icon URL**:

  The URL to your asset's icon (which will be used as a thumbnail in the AssetLib search results and on the asset's page). Should be an image in either the PNG or JPG format.

  The **icon** must be square (1:1 aspect ratio). It should have a minimum resolution of 128×128 pixels.

Note

For icons hosted on GitHub, URLs must be provided in the form of *https://raw.githubusercontent.com/<user>/<project>/<branch>/Icon.png*.

- **License**:
  The license under which you are distributing the asset. The list includes a variety of free and open source software licenses, such as GPL (v2 and v3), MIT, BSD and Boost Software License. You can visit OpenSource.org [https://opensource.org] for a detailed description of each of the listed licenses.
- **Description**:
  Finally, you can use the Description field for a textual overview of your asset, its features and behavior, a changelog, et cetera. In the future, formatting with Markdown will be supported, but currently, your only option is plain text.

You may also include up to three video and/or image previews, which will be shown at the bottom of the asset page. Use the "Enable" checkbox on each of the preview submission boxes to enable them.

- **Type**:
  Either an image, or a video.
- **Image/YouTube URL**:
  Either a link to the image, or to a video, hosted on YouTube.
- **Thumbnail URL**:
  A URL to an image that will be used as a thumbnail for the preview. This option will be removed eventually, and thumbnails will be automatically computed instead.

Once you are done, press "Submit". Your asset will be entered into the review queue. You can check all assets currently pending a review here [https://godotengine.org/asset-library/asset/edit?&asset=-1] . The approval process is manual and may take up to a few days for your asset to be accepted (or rejected), so please be patient!

Note

You may have some luck accelerating the approval process by messaging the moderators and AssetLib reviewers on the Godot Contributors Chat [https://chat.godotengine.org/], or the official Discord server.

You will be informed when your asset is reviewed. If it was rejected, you will be told why that may have been, and you will be able to submit it again with the appropriate changes.

Community channels

So, where is the Godot community and where can you ask questions and get help?

This page used to list the various official and user-supported Godot communities. That list is now available on the Godot website [https://godotengine.org/community].

Language-based communities

See the User groups [https://godotengine.org/community/user-groups] page of the website for a list of local communities.

Tutorials and resources

This is a list of third-party tutorials and resources created by the Godot community. For resources, remember that there is the official Godot Asset Library [https://godotengine.org/asset-library/asset] full of official and community resources too! Also, have a look at this huge list over at Reddit [https://www.reddit.com/r/godot/comments/an0iq5/godot_tutorials_list_of_video_and_written/].

Think there is something missing here? Feel free to submit a Pull Request [https://github.com/godotengine/godot-docs/blob/master/community/tutorials.rst] as always.

Where to start

The Godot video tutorials by GDQuest [https://www.youtube.com/channel/UCxboW7x0jZqFdvMdCFKTMsQ/playlists] are well-regarded in the community and often recommended as a gentle introduction to beginners.

GDQuest's *Learn GDScript From Zero* is a free and open source interactive tutorial for absolute beginners to learn to program with Godot's GDScript language. It is available as a desktop application [https://gdquest.itch.io/learn-godot-gdscript] or in the browser [https://gdquest.github.io/learn-gdscript].

Some tutorials mentioned below cover more advanced subjects, e.g. on 3D or shaders.

Video tutorials

For video tutorials we recommend looking on YouTube [https://www.youtube.com/]. There's many great channels covering a wide array of subjects.

Text tutorials

- FinePointCGI website by Mitch [https://finepointcgi.io/]
- GDScript website by Andrew Wilkes [https://gdscript.com]
- Godot Recipes by KidsCanCode [https://kidscancode.org/godot_recipes/4.x/]
- Godot Tutorials by SomethingLikeGames [https://www.somethinglikegames.de/en/tags/godot-engine/]
- Game Dev Artisan website [https://gamedevartisan.com/]
- Night Quest Games Blog [https://www.nightquestgames.com/blog-articles/]

Devlogs

- bitbrain [https://www.youtube.com/@bitbraindev]
- DevDuck (2D) [https://www.youtube.com/@devduck/videos]

Resources

- awesome-godot: A curated list of free/libre plugins, scripts and add-ons [https://github.com/godotengine/awesome-godot]
- Godot Asset Library [https://godotengine.org/asset-library/asset]
- Godot Shaders: A community-driven shader library [https://godotshaders.com/]

All classes

Globals

- @GDScript
- @GlobalScope

Nodes

- Node
- AcceptDialog
- AnimatableBody2D
- AnimatableBody3D
- AnimatedSprite2D
- AnimatedSprite3D
- AnimationMixer
- AnimationPlayer
- AnimationTree
- Area2D
- Area3D
- AspectRatioContainer
- AudioListener2D
- AudioListener3D
- AudioStreamPlayer
- AudioStreamPlayer2D
- AudioStreamPlayer3D
- BackBufferCopy
- BaseButton
- Bone2D
- BoneAttachment3D
- BoxContainer

- Button
- Camera2D
- Camera3D
- CanvasGroup
- CanvasItem
- CanvasLayer
- CanvasModulate
- CenterContainer
- CharacterBody2D
- CharacterBody3D
- CheckBox
- CheckButton
- CodeEdit
- CollisionObject2D
- CollisionObject3D
- CollisionPolygon2D
- CollisionPolygon3D
- CollisionShape2D
- CollisionShape3D
- ColorPicker
- ColorPickerButton
- ColorRect
- ConeTwistJoint3D
- ConfirmationDialog
- Container
- Control
- CPUParticles2D
- CPUParticles3D
- CSGBox3D
- CSGCombiner3D
- CSGCylinder3D
- CSGMesh3D
- CSGPolygon3D
- CSGPrimitive3D
- CSGShape3D
- CSGSphere3D

- CSGTorus3D
- DampedSpringJoint2D
- Decal
- DirectionalLight2D
- DirectionalLight3D
- EditorCommandPalette
- EditorFileDialog
- EditorFileSystem
- EditorInspector
- EditorPlugin
- EditorProperty
- EditorResourcePicker
- EditorResourcePreview
- EditorScriptPicker
- EditorSpinSlider
- FileDialog
- FileSystemDock
- FlowContainer
- FogVolume
- Generic6DOFJoint3D
- GeometryInstance3D
- GPUParticles2D
- GPUParticles3D
- GPUParticlesAttractor3D
- GPUParticlesAttractorBox3D
- GPUParticlesAttractorSphere3D
- GPUParticlesAttractorVectorField3D
- GPUParticlesCollision3D
- GPUParticlesCollisionBox3D
- GPUParticlesCollisionHeightField3D
- GPUParticlesCollisionSDF3D
- GPUParticlesCollisionSphere3D
- GraphEdit
- GraphElement
- GraphFrame
- GraphNode

- GridContainer
- GridMap
- GrooveJoint2D
- HBoxContainer
- HFlowContainer
- HingeJoint3D
- HScrollBar
- HSeparator
- HSlider
- HSplitContainer
- HTTPRequest
- ImporterMeshInstance3D
- InstancePlaceholder
- ItemList
- Joint2D
- Joint3D
- Label
- Label3D
- Light2D
- Light3D
- LightmapGI
- LightmapProbe
- LightOccluder2D
- Line2D
- LineEdit
- LinkButton
- MarginContainer
- Marker2D
- Marker3D
- MenuBar
- MenuButton
- MeshInstance2D
- MeshInstance3D
- MissingNode
- MultiMeshInstance2D
- MultiMeshInstance3D

- MultiplayerSpawner
- MultiplayerSynchronizer
- NavigationAgent2D
- NavigationAgent3D
- NavigationLink2D
- NavigationLink3D
- NavigationObstacle2D
- NavigationObstacle3D
- NavigationRegion2D
- NavigationRegion3D
- NinePatchRect
- Node2D
- Node3D
- OccluderInstance3D
- OmniLight3D
- OpenXRCompositionLayer
- OpenXRCompositionLayerCylinder
- OpenXRCompositionLayerEquirect
- OpenXRCompositionLayerQuad
- OpenXRHand
- OptionButton
- Panel
- PanelContainer
- Parallax2D
- ParallaxBackground
- ParallaxLayer
- Path2D
- Path3D
- PathFollow2D
- PathFollow3D
- PhysicalBone2D
- PhysicalBone3D
- PhysicalBoneSimulator3D
- PhysicsBody2D
- PhysicsBody3D
- PinJoint2D

- PinJoint3D
- PointLight2D
- Polygon2D
- Popup
- PopupMenu
- PopupPanel
- ProgressBar
- Range
- RayCast2D
- RayCast3D
- ReferenceRect
- ReflectionProbe
- RemoteTransform2D
- RemoteTransform3D
- ResourcePreloader
- RichTextLabel
- RigidBody2D
- RigidBody3D
- RootMotionView
- ScriptCreateDialog
- ScriptEditor
- ScriptEditorBase
- ScrollBar
- ScrollContainer
- Separator
- ShaderGlobalsOverride
- ShapeCast2D
- ShapeCast3D
- Skeleton2D
- Skeleton3D
- SkeletonIK3D
- SkeletonModifier3D
- Slider
- SliderJoint3D
- SoftBody3D
- SpinBox

- SplitContainer
- SpotLight3D
- SpringArm3D
- Sprite2D
- Sprite3D
- SpriteBase3D
- StaticBody2D
- StaticBody3D
- StatusIndicator
- SubViewport
- SubViewportContainer
- TabBar
- TabContainer
- TextEdit
- TextureButton
- TextureProgressBar
- TextureRect
- TileMap
- TileMapLayer
- Timer
- TouchScreenButton
- Tree
- VBoxContainer
- VehicleBody3D
- VehicleWheel3D
- VFlowContainer
- VideoStreamPlayer
- Viewport
- VisibleOnScreenEnabler2D
- VisibleOnScreenEnabler3D
- VisibleOnScreenNotifier2D
- VisibleOnScreenNotifier3D
- VisualInstance3D
- VoxelGI
- VScrollBar
- VSeparator

- VSlider
- VSplitContainer
- Window
- WorldEnvironment
- XRAnchor3D
- XRBodyModifier3D
- XRCamera3D
- XRController3D
- XRFaceModifier3D
- XRHandModifier3D
- XRNode3D
- XROrigin3D

Resources

- Resource
- AnimatedTexture
- Animation
- AnimationLibrary
- AnimationNode
- AnimationNodeAdd2
- AnimationNodeAdd3
- AnimationNodeAnimation
- AnimationNodeBlend2
- AnimationNodeBlend3
- AnimationNodeBlendSpace1D
- AnimationNodeBlendSpace2D
- AnimationNodeBlendTree
- AnimationNodeOneShot
- AnimationNodeOutput
- AnimationNodeStateMachine
- AnimationNodeStateMachinePlayback
- AnimationNodeStateMachineTransition
- AnimationNodeSub2

- AnimationNodeSync
- AnimationNodeTimeScale
- AnimationNodeTimeSeek
- AnimationNodeTransition
- AnimationRootNode
- ArrayMesh
- ArrayOccluder3D
- AtlasTexture
- AudioBusLayout
- AudioEffect
- AudioEffectAmplify
- AudioEffectBandLimitFilter
- AudioEffectBandPassFilter
- AudioEffectCapture
- AudioEffectChorus
- AudioEffectCompressor
- AudioEffectDelay
- AudioEffectDistortion
- AudioEffectEQ
- AudioEffectEQ10
- AudioEffectEQ21
- AudioEffectEQ6
- AudioEffectFilter
- AudioEffectHardLimiter
- AudioEffectHighPassFilter
- AudioEffectHighShelfFilter
- AudioEffectLimiter
- AudioEffectLowPassFilter
- AudioEffectLowShelfFilter
- AudioEffectNotchFilter
- AudioEffectPanner
- AudioEffectPhaser
- AudioEffectPitchShift
- AudioEffectRecord
- AudioEffectReverb
- AudioEffectSpectrumAnalyzer

- AudioEffectStereoEnhance
- AudioStream
- AudioStreamGenerator
- AudioStreamInteractive
- AudioStreamMicrophone
- AudioStreamMP3
- AudioStreamOggVorbis
- AudioStreamPlaylist
- AudioStreamPolyphonic
- AudioStreamRandomizer
- AudioStreamSynchronized
- AudioStreamWAV
- BaseMaterial3D
- BitMap
- BoneMap
- BoxMesh
- BoxOccluder3D
- BoxShape3D
- ButtonGroup
- CameraAttributes
- CameraAttributesPhysical
- CameraAttributesPractical
- CameraTexture
- CanvasItemMaterial
- CanvasTexture
- CapsuleMesh
- CapsuleShape2D
- CapsuleShape3D
- CircleShape2D
- CodeHighlighter
- Compositor
- CompositorEffect
- CompressedCubemap
- CompressedCubemapArray
- CompressedTexture2D
- CompressedTexture2DArray

- CompressedTexture3D
- CompressedTextureLayered
- ConcavePolygonShape2D
- ConcavePolygonShape3D
- ConvexPolygonShape2D
- ConvexPolygonShape3D
- CryptoKey
- CSharpScript
- Cubemap
- CubemapArray
- Curve
- Curve2D
- Curve3D
- CurveTexture
- CurveXYZTexture
- CylinderMesh
- CylinderShape3D
- EditorNode3DGizmoPlugin
- EditorSettings
- EditorSyntaxHighlighter
- Environment
- FastNoiseLite
- FBXDocument
- FBXState
- FogMaterial
- Font
- FontFile
- FontVariation
- GDExtension
- GDScript
- GLTFAccessor
- GLTFAnimation
- GLTFBufferView
- GLTFCamera
- GLTFDocument
- GLTFDocumentExtension

- GLTFDocumentExtensionConvertImporterMesh
- GLTFLight
- GLTFMesh
- GLTFNode
- GLTFPhysicsBody
- GLTFPhysicsShape
- GLTFSkeleton
- GLTFSkin
- GLTFSpecGloss
- GLTFState
- GLTFTexture
- GLTFTextureSampler
- Gradient
- GradientTexture1D
- GradientTexture2D
- HeightMapShape3D
- Image
- ImageTexture
- ImageTexture3D
- ImageTextureLayered
- ImmediateMesh
- ImporterMesh
- InputEvent
- InputEventAction
- InputEventFromWindow
- InputEventGesture
- InputEventJoypadButton
- InputEventJoypadMotion
- InputEventKey
- InputEventMagnifyGesture
- InputEventMIDI
- InputEventMouse
- InputEventMouseButton
- InputEventMouseMotion
- InputEventPanGesture
- InputEventScreenDrag

- InputEventScreenTouch
- InputEventShortcut
- InputEventWithModifiers
- JSON
- LabelSettings
- LightmapGIData
- Material
- Mesh
- MeshLibrary
- MeshTexture
- MissingResource
- MultiMesh
- NavigationMesh
- NavigationMeshSourceGeometryData2D
- NavigationMeshSourceGeometryData3D
- NavigationPolygon
- Noise
- NoiseTexture2D
- NoiseTexture3D
- Occluder3D
- OccluderPolygon2D
- OggPacketSequence
- OpenXRAction
- OpenXRActionMap
- OpenXRActionSet
- OpenXRInteractionProfile
- OpenXRIPBinding
- OptimizedTranslation
- ORMMaterial3D
- PackedDataContainer
- PackedScene
- PanoramaSkyMaterial
- ParticleProcessMaterial
- PhysicalSkyMaterial
- PhysicsMaterial
- PlaceholderCubemap

- PlaceholderCubemapArray
- PlaceholderMaterial
- PlaceholderMesh
- PlaceholderTexture2D
- PlaceholderTexture2DArray
- PlaceholderTexture3D
- PlaceholderTextureLayered
- PlaneMesh
- PointMesh
- PolygonOccluder3D
- PolygonPathFinder
- PortableCompressedTexture2D
- PrimitiveMesh
- PrismMesh
- ProceduralSkyMaterial
- QuadMesh
- QuadOccluder3D
- RDShaderFile
- RDShaderSPIRV
- RectangleShape2D
- RibbonTrailMesh
- RichTextEffect
- SceneReplicationConfig
- Script
- ScriptExtension
- SegmentShape2D
- SeparationRayShape2D
- SeparationRayShape3D
- Shader
- ShaderInclude
- ShaderMaterial
- Shape2D
- Shape3D
- Shortcut
- SkeletonModification2D
- SkeletonModification2DCCDIK

- SkeletonModification2DFABRIK
- SkeletonModification2DJiggle
- SkeletonModification2DLookAt
- SkeletonModification2DPhysicalBones
- SkeletonModification2DStackHolder
- SkeletonModification2DTwoBoneIK
- SkeletonModificationStack2D
- SkeletonProfile
- SkeletonProfileHumanoid
- Skin
- Sky
- SphereMesh
- SphereOccluder3D
- SphereShape3D
- SpriteFrames
- StandardMaterial3D
- StyleBox
- StyleBoxEmpty
- StyleBoxFlat
- StyleBoxLine
- StyleBoxTexture
- SyntaxHighlighter
- SystemFont
- TextMesh
- Texture
- Texture2D
- Texture2DArray
- Texture2DArrayRD
- Texture2DRD
- Texture3D
- Texture3DRD
- TextureCubemapArrayRD
- TextureCubemapRD
- TextureLayered
- TextureLayeredRD
- Theme

- TileMapPattern
- TileSet
- TileSetAtlasSource
- TileSetScenesCollectionSource
- TileSetSource
- TorusMesh
- Translation
- TubeTrailMesh
- VideoStream
- VideoStreamPlayback
- VideoStreamTheora
- ViewportTexture
- VisualShader
- VisualShaderNode
- VisualShaderNodeBillboard
- VisualShaderNodeBooleanConstant
- VisualShaderNodeBooleanParameter
- VisualShaderNodeClamp
- VisualShaderNodeColorConstant
- VisualShaderNodeColorFunc
- VisualShaderNodeColorOp
- VisualShaderNodeColorParameter
- VisualShaderNodeComment
- VisualShaderNodeCompare
- VisualShaderNodeConstant
- VisualShaderNodeCubemap
- VisualShaderNodeCubemapParameter
- VisualShaderNodeCurveTexture
- VisualShaderNodeCurveXYZTexture
- VisualShaderNodeCustom
- VisualShaderNodeDerivativeFunc
- VisualShaderNodeDeterminant
- VisualShaderNodeDistanceFade
- VisualShaderNodeDotProduct
- VisualShaderNodeExpression
- VisualShaderNodeFaceForward

- VisualShaderNodeFloatConstant
- VisualShaderNodeFloatFunc
- VisualShaderNodeFloatOp
- VisualShaderNodeFloatParameter
- VisualShaderNodeFrame
- VisualShaderNodeFresnel
- VisualShaderNodeGlobalExpression
- VisualShaderNodeGroupBase
- VisualShaderNodeIf
- VisualShaderNodeInput
- VisualShaderNodeIntConstant
- VisualShaderNodeIntFunc
- VisualShaderNodeIntOp
- VisualShaderNodeIntParameter
- VisualShaderNodeIs
- VisualShaderNodeLinearSceneDepth
- VisualShaderNodeMix
- VisualShaderNodeMultiplyAdd
- VisualShaderNodeOuterProduct
- VisualShaderNodeOutput
- VisualShaderNodeParameter
- VisualShaderNodeParameterRef
- VisualShaderNodeParticleAccelerator
- VisualShaderNodeParticleBoxEmitter
- VisualShaderNodeParticleConeVelocity
- VisualShaderNodeParticleEmit
- VisualShaderNodeParticleEmitter
- VisualShaderNodeParticleMeshEmitter
- VisualShaderNodeParticleMultiplyByAxisAngle
- VisualShaderNodeParticleOutput
- VisualShaderNodeParticleRandomness
- VisualShaderNodeParticleRingEmitter
- VisualShaderNodeParticleSphereEmitter
- VisualShaderNodeProximityFade
- VisualShaderNodeRandomRange
- VisualShaderNodeRemap

- VisualShaderNodeReroute
- VisualShaderNodeResizableBase
- VisualShaderNodeRotationByAxis
- VisualShaderNodeSample3D
- VisualShaderNodeScreenNormalWorldSpace
- VisualShaderNodeScreenUVToSDF
- VisualShaderNodeSDFRaymarch
- VisualShaderNodeSDFToScreenUV
- VisualShaderNodeSmoothStep
- VisualShaderNodeStep
- VisualShaderNodeSwitch
- VisualShaderNodeTexture
- VisualShaderNodeTexture2DArray
- VisualShaderNodeTexture2DArrayParameter
- VisualShaderNodeTexture2DParameter
- VisualShaderNodeTexture3D
- VisualShaderNodeTexture3DParameter
- VisualShaderNodeTextureParameter
- VisualShaderNodeTextureParameterTriplanar
- VisualShaderNodeTextureSDF
- VisualShaderNodeTextureSDFNormal
- VisualShaderNodeTransformCompose
- VisualShaderNodeTransformConstant
- VisualShaderNodeTransformDecompose
- VisualShaderNodeTransformFunc
- VisualShaderNodeTransformOp
- VisualShaderNodeTransformParameter
- VisualShaderNodeTransformVecMult
- VisualShaderNodeUIntConstant
- VisualShaderNodeUIntFunc
- VisualShaderNodeUIntOp
- VisualShaderNodeUIntParameter
- VisualShaderNodeUVFunc
- VisualShaderNodeUVPolarCoord
- VisualShaderNodeVarying
- VisualShaderNodeVaryingGetter

- VisualShaderNodeVaryingSetter
- VisualShaderNodeVec2Constant
- VisualShaderNodeVec2Parameter
- VisualShaderNodeVec3Constant
- VisualShaderNodeVec3Parameter
- VisualShaderNodeVec4Constant
- VisualShaderNodeVec4Parameter
- VisualShaderNodeVectorBase
- VisualShaderNodeVectorCompose
- VisualShaderNodeVectorDecompose
- VisualShaderNodeVectorDistance
- VisualShaderNodeVectorFunc
- VisualShaderNodeVectorLen
- VisualShaderNodeVectorOp
- VisualShaderNodeVectorRefract
- VisualShaderNodeWorldPositionFromDepth
- VoxelGIData
- World2D
- World3D
- WorldBoundaryShape2D
- WorldBoundaryShape3D
- X509Certificate

Other objects

- Object
- AESContext
- AStar2D
- AStar3D
- AStarGrid2D
- AudioEffectInstance
- AudioEffectSpectrumAnalyzerInstance
- AudioSample
- AudioSamplePlayback

- AudioServer
- AudioStreamGeneratorPlayback
- AudioStreamPlayback
- AudioStreamPlaybackInteractive
- AudioStreamPlaybackOggVorbis
- AudioStreamPlaybackPlaylist
- AudioStreamPlaybackPolyphonic
- AudioStreamPlaybackResampled
- AudioStreamPlaybackSynchronized
- CallbackTweener
- CameraFeed
- CameraServer
- CharFXTransform
- ClassDB
- ConfigFile
- Crypto
- DirAccess
- DisplayServer
- DTLSServer
- EditorDebuggerPlugin
- EditorDebuggerSession
- EditorExportPlatform
- EditorExportPlatformAndroid
- EditorExportPlatformIOS
- EditorExportPlatformLinuxBSD
- EditorExportPlatformMacOS
- EditorExportPlatformPC
- EditorExportPlatformWeb
- EditorExportPlatformWindows
- EditorExportPlugin
- EditorFeatureProfile
- EditorFileSystemDirectory
- EditorFileSystemImportFormatSupportQuery
- EditorImportPlugin
- EditorInspectorPlugin
- EditorInterface

- EditorNode3DGizmo
- EditorPaths
- EditorResourceConversionPlugin
- EditorResourcePreviewGenerator
- EditorResourceTooltipPlugin
- EditorSceneFormatImporter
- EditorSceneFormatImporterBlend
- EditorSceneFormatImporterFBX2GLTF
- EditorSceneFormatImporterGLTF
- EditorSceneFormatImporterUFBX
- EditorScenePostImport
- EditorScenePostImportPlugin
- EditorScript
- EditorSelection
- EditorTranslationParserPlugin
- EditorUndoRedoManager
- EditorVCSInterface
- EncodedObjectAsID
- ENetConnection
- ENetMultiplayerPeer
- ENetPacketPeer
- Engine
- EngineDebugger
- EngineProfiler
- Expression
- FileAccess
- FramebufferCacheRD
- GDExtensionManager
- Geometry2D
- Geometry3D
- HashingContext
- HMACContext
- HTTPClient
- ImageFormatLoader
- ImageFormatLoaderExtension
- Input

- InputMap
- IntervalTweener
- IP
- JavaClass
- JavaClassWrapper
- JavaScriptBridge
- JavaScriptObject
- JNISingleton
- JSONRPC
- KinematicCollision2D
- KinematicCollision3D
- Lightmapper
- LightmapperRD
- MainLoop
- Marshalls
- MeshConvexDecompositionSettings
- MeshDataTool
- MethodTweener
- MobileVRInterface
- MovieWriter
- MultiplayerAPI
- MultiplayerAPIExtension
- MultiplayerPeer
- MultiplayerPeerExtension
- Mutex
- NativeMenu
- NavigationMeshGenerator
- NavigationPathQueryParameters2D
- NavigationPathQueryParameters3D
- NavigationPathQueryResult2D
- NavigationPathQueryResult3D
- NavigationServer2D
- NavigationServer3D
- Node
- Node3DGizmo
- OfflineMultiplayerPeer

- OggPacketSequencePlayback
- OpenXRAPIExtension
- OpenXRExtensionWrapperExtension
- OpenXRInteractionProfileMetadata
- OpenXRInterface
- OS
- PackedDataContainerRef
- PacketPeer
- PacketPeerDTLS
- PacketPeerExtension
- PacketPeerStream
- PacketPeerUDP
- PCKPacker
- Performance
- PhysicsDirectBodyState2D
- PhysicsDirectBodyState2DExtension
- PhysicsDirectBodyState3D
- PhysicsDirectBodyState3DExtension
- PhysicsDirectSpaceState2D
- PhysicsDirectSpaceState2DExtension
- PhysicsDirectSpaceState3D
- PhysicsDirectSpaceState3DExtension
- PhysicsPointQueryParameters2D
- PhysicsPointQueryParameters3D
- PhysicsRayQueryParameters2D
- PhysicsRayQueryParameters3D
- PhysicsServer2D
- PhysicsServer2DExtension
- PhysicsServer2DManager
- PhysicsServer3D
- PhysicsServer3DExtension
- PhysicsServer3DManager
- PhysicsServer3DRenderingServerHandler
- PhysicsShapeQueryParameters2D
- PhysicsShapeQueryParameters3D
- PhysicsTestMotionParameters2D

- PhysicsTestMotionParameters3D
- PhysicsTestMotionResult2D
- PhysicsTestMotionResult3D
- ProjectSettings
- PropertyTweener
- RandomNumberGenerator
- RDAttachmentFormat
- RDFramebufferPass
- RDPipelineColorBlendState
- RDPipelineColorBlendStateAttachment
- RDPipelineDepthStencilState
- RDPipelineMultisampleState
- RDPipelineRasterizationState
- RDPipelineSpecializationConstant
- RDSamplerState
- RDShaderSource
- RDTextureFormat
- RDTextureView
- RDUniform
- RDVertexAttribute
- RefCounted
- RegEx
- RegExMatch
- RenderData
- RenderDataExtension
- RenderDataRD
- RenderingDevice
- RenderingServer
- RenderSceneBuffers
- RenderSceneBuffersConfiguration
- RenderSceneBuffersExtension
- RenderSceneBuffersRD
- RenderSceneData
- RenderSceneDataExtension
- RenderSceneDataRD
- Resource

- ResourceFormatLoader
- ResourceFormatSaver
- ResourceImporter
- ResourceImporterBitMap
- ResourceImporterBMFont
- ResourceImporterCSVTranslation
- ResourceImporterDynamicFont
- ResourceImporterImage
- ResourceImporterImageFont
- ResourceImporterLayeredTexture
- ResourceImporterMP3
- ResourceImporterOBJ
- ResourceImporterOggVorbis
- ResourceImporterScene
- ResourceImporterShaderFile
- ResourceImporterTexture
- ResourceImporterTextureAtlas
- ResourceImporterWAV
- ResourceLoader
- ResourceSaver
- ResourceUID
- SceneMultiplayer
- SceneState
- SceneTree
- SceneTreeTimer
- ScriptLanguage
- ScriptLanguageExtension
- Semaphore
- SkinReference
- StreamPeer
- StreamPeerBuffer
- StreamPeerExtension
- StreamPeerGZIP
- StreamPeerTCP
- StreamPeerTLS
- SurfaceTool

- TCPServer
- TextLine
- TextParagraph
- TextServer
- TextServerAdvanced
- TextServerDummy
- TextServerExtension
- TextServerFallback
- TextServerManager
- ThemeDB
- Thread
- TileData
- Time
- TLSOptions
- TranslationServer
- TreeItem
- TriangleMesh
- Tween
- Tweener
- UDPServer
- UndoRedo
- UniformSetCacheRD
- UPNP
- UPNPDevice
- WeakRef
- WebRTCDataChannel
- WebRTCDataChannelExtension
- WebRTCMultiplayerPeer
- WebRTCPeerConnection
- WebRTCPeerConnectionExtension
- WebSocketMultiplayerPeer
- WebSocketPeer
- WebXRInterface
- WorkerThreadPool
- XMLParser
- XRBodyTracker

- XRControllerTracker
- XRFaceTracker
- XRHandTracker
- XRInterface
- XRInterfaceExtension
- XRPose
- XRPositionalTracker
- XRServer
- XRTracker
- XRVRS
- ZIPPacker
- ZIPReader

Editor-only

- EditorCommandPalette
- EditorDebuggerPlugin
- EditorDebuggerSession
- EditorExportPlatform
- EditorExportPlatformAndroid
- EditorExportPlatformIOS
- EditorExportPlatformLinuxBSD
- EditorExportPlatformMacOS
- EditorExportPlatformPC
- EditorExportPlatformWeb
- EditorExportPlatformWindows
- EditorExportPlugin
- EditorFeatureProfile
- EditorFileDialog
- EditorFileSystem
- EditorFileSystemDirectory
- EditorFileSystemImportFormatSupportQuery
- EditorImportPlugin
- EditorInspector

- EditorInspectorPlugin
- EditorInterface
- EditorNode3DGizmo
- EditorNode3DGizmoPlugin
- EditorPaths
- EditorPlugin
- EditorProperty
- EditorResourceConversionPlugin
- EditorResourcePicker
- EditorResourcePreview
- EditorResourcePreviewGenerator
- EditorResourceTooltipPlugin
- EditorSceneFormatImporter
- EditorSceneFormatImporterBlend
- EditorSceneFormatImporterFBX2GLTF
- EditorSceneFormatImporterGLTF
- EditorSceneFormatImporterUFBX
- EditorScenePostImport
- EditorScenePostImportPlugin
- EditorScript
- EditorScriptPicker
- EditorSelection
- EditorSettings
- EditorSpinSlider
- EditorSyntaxHighlighter
- EditorTranslationParserPlugin
- EditorUndoRedoManager
- EditorVCSInterface
- FileSystemDock
- ScriptCreateDialog
- ScriptEditor
- ScriptEditorBase

Variant types

- Variant
- AABB
- Array
- Basis
- bool
- Callable
- Color
- Dictionary
- float
- int
- NodePath
- Object
- PackedByteArray
- PackedColorArray
- PackedFloat32Array
- PackedFloat64Array
- PackedInt32Array
- PackedInt64Array
- PackedStringArray
- PackedVector2Array
- PackedVector3Array
- PackedVector4Array
- Plane
- Projection
- Quaternion
- Rect2
- Rect2i
- RID
- Signal
- String
- StringName
- Transform2D
- Transform3D
- Vector2
- Vector2i
- Vector3

- Vector3i
- Vector4
- Vector4i

Node

Inherits: Object

Inherited By: AnimationMixer, AudioStreamPlayer, CanvasItem, CanvasLayer, EditorFileSystem, EditorPlugin, EditorResourcePreview, HTTPRequest, InstancePlaceholder, MissingNode, MultiplayerSpawner, MultiplayerSynchronizer, NavigationAgent2D, NavigationAgent3D, Node3D, ResourcePreloader, ShaderGlobalsOverride, StatusIndicator, Timer, Viewport, WorldEnvironment

Base class for all scene objects.

Description

Nodes are Godot's building blocks. They can be assigned as the child of another node, resulting in a tree arrangement. A given node can contain any number of nodes as children with the requirement that all siblings (direct children of a node) should have unique names.

A tree of nodes is called a *scene*. Scenes can be saved to the disk and then instantiated into other scenes. This allows for very high flexibility in the architecture and data model of Godot projects.

Scene tree: The SceneTree contains the active tree of nodes. When a node is added to the scene tree, it receives the NOTIFICATION_ENTER_TREE notification and its _enter_tree callback is triggered. Child nodes are always added *after* their parent node, i.e. the _enter_tree callback of a parent node will be triggered before its child's.

Once all nodes have been added in the scene tree, they receive the NOTIFICATION_READY notification and their respective _ready callbacks are triggered. For groups of nodes, the _ready callback is called in reverse order, starting with the children and moving up to the parent nodes.

This means that when adding a node to the scene tree, the following order will be used for the callbacks: _enter_tree of the parent, _enter_tree of the children, _ready of the children and finally _ready of the parent (recursively for the entire scene tree).

Processing: Nodes can override the "process" state, so that they receive a callback on each frame requesting them to process (do something). Normal processing (callback _process, toggled with set_process) happens as fast as possible and is dependent on the frame rate, so the processing time *delta* (in seconds) is passed as an argument. Physics processing (callback _physics_process, toggled with set_physics_process) happens a fixed number of times per second (60 by default) and is useful for code related to the physics engine.

Nodes can also process input events. When present, the _input function will be called for each input that the program receives. In many cases, this can be overkill (unless used for simple projects), and the _unhandled_input function might be preferred; it is called when the input event was not handled by anyone else (typically, GUI Control nodes), ensuring that the node only receives the events that were meant for it.

To keep track of the scene hierarchy (especially when instantiating scenes into other scenes), an "owner" can be set for the node with the owner property. This keeps track of who instantiated what. This is mostly useful when writing editors and tools, though.

Finally, when a node is freed with Object.free or queue_free, it will also free all its children.

Groups: Nodes can be added to as many groups as you want to be easy to manage, you could create groups like "enemies" or "collectables" for example, depending on your game. See add_to_group, is_in_group and remove_from_group. You can then retrieve all nodes in these groups, iterate them and even call methods on groups via the methods on SceneTree.

Networking with nodes: After connecting to a server (or making one, see ENetMultiplayerPeer), it is possible to use the built-in RPC (remote procedure call) system to communicate over the network. By calling rpc with a method name, it will be called locally and in all connected peers (peers = clients and the server that accepts connections). To identify which node receives the RPC call, Godot will use its NodePath (make sure node names are the same on all peers). Also, take a look at the high-level networking tutorial and corresponding demos.

Note: The `script` property is part of the Object class, not **Node**. It isn't exposed like most properties but does have a setter and getter (see Object.set_script and Object.get_script).

Tutorials

- Nodes and scenes
- All Demos [https://github.com/godotengine/godot-demo-projects/]

Properties

AutoTranslateMode	auto_translate_mode	`0`
String	editor_description	`""`
MultiplayerAPI	multiplayer	
StringName	name	
Node	owner	
PhysicsInterpolationMode	physics_interpolation_mode	`0`
ProcessMode	process_mode	`0`
int	process_physics_priority	`0`

int	process_priority	`0`
ProcessThreadGroup	process_thread_group	`0`
int	process_thread_group_order	
BitField[ProcessThreadMessages]	process_thread_messages	
String	scene_file_path	
bool	unique_name_in_owner	`false`

Methods

void	_enter_tree() virtual
void	_exit_tree() virtual
PackedStringArray	_get_configuration_warnings() virtual const
void	_input(event: InputEvent) virtual
void	_physics_process(delta: float) virtual
void	_process(delta: float) virtual
void	_ready() virtual

void	_shortcut_input(event: InputEvent) virtual
void	_unhandled_input(event: InputEvent) virtual
void	_unhandled_key_input(event: InputEvent) virtual
void	add_child(node: Node, force_readable_name: bool = false, internal: InternalMode = 0)
void	add_sibling(sibling: Node, force_readable_name: bool = false)
void	add_to_group(group: StringName, persistent: bool = false)
String	atr(message: String, context: StringName = "") const
String	atr_n(message: String, plural_message: StringName, n: int, context: StringName = "") const
Variant	call_deferred_thread_group(method: StringName, ...) vararg
Variant	call_thread_safe(method: StringName, ...) vararg
bool	can_process() const
Tween	create_tween()

Node	duplicate(flags: int = 15) const
Node	find_child(pattern: String, recursive: bool = true, owned: bool = true) const
Array[Node]	find_children(pattern: String, type: String = "", recursive: bool = true, owned: bool = true) const
Node	find_parent(pattern: String) const
Node	get_child(idx: int, include_internal: bool = false) const
int	get_child_count(include_internal: bool = false) const
Array[Node]	get_children(include_internal: bool = false) const
Array[StringName]	get_groups() const
int	get_index(include_internal: bool = false) const
Window	get_last_exclusive_window() const
int	get_multiplayer_authority() const
Node	get_node(path: NodePath) const
Array	get_node_and_resource(path: NodePath)

Node	get_node_or_null(path: NodePath) const
Node	get_parent() const
NodePath	get_path() const
NodePath	get_path_to(node: Node, use_unique_path: bool = false) const
float	get_physics_process_delta_time() const
float	get_process_delta_time() const
bool	get_scene_instance_load_placeholder() const
SceneTree	get_tree() const
String	get_tree_string()
String	get_tree_string_pretty()
Viewport	get_viewport() const
Window	get_window() const
bool	has_node(path: NodePath) const
bool	has_node_and_resource(path: NodePath) const

bool	is_ancestor_of(node: Node) const
bool	is_displayed_folded() const
bool	is_editable_instance(node: Node) const
bool	is_greater_than(node: Node) const
bool	is_in_group(group: StringName) const
bool	is_inside_tree() const
bool	is_multiplayer_authority() const
bool	is_node_ready() const
bool	is_part_of_edited_scene() const
bool	is_physics_interpolated() const
bool	is_physics_interpolated_and_enabled() const
bool	is_physics_processing() const
bool	is_physics_processing_internal() const
bool	is_processing() const

bool	is_processing_input() const
bool	is_processing_internal() const
bool	is_processing_shortcut_input() const
bool	is_processing_unhandled_input() const
bool	is_processing_unhandled_key_input() const
void	move_child(child_node: Node, to_index: int)
void	notify_deferred_thread_group(what: int)
void	notify_thread_safe(what: int)
void	print_orphan_nodes() static
void	print_tree()
void	print_tree_pretty()
void	propagate_call(method: StringName, args: Array = [], parent_first: bool = false)
void	propagate_notification(what: int)
void	queue_free()

void	remove_child(node: Node)
void	remove_from_group(group: StringName)
void	reparent(new_parent: Node, keep_global_transform: bool = true)
void	replace_by(node: Node, keep_groups: bool = false)
void	request_ready()
void	reset_physics_interpolation()
Error	rpc(method: StringName, ...) vararg
void	rpc_config(method: StringName, config: Variant)
Error	rpc_id(peer_id: int, method: StringName, ...) vararg
void	set_deferred_thread_group(property: StringName, value: Variant)
void	set_display_folded(fold: bool)
void	set_editable_instance(node: Node, is_editable: bool)
void	set_multiplayer_authority(id: int, recursive: bool = true)

void	set_physics_process(enable: bool)
void	set_physics_process_internal(enable: bool)
void	set_process(enable: bool)
void	set_process_input(enable: bool)
void	set_process_internal(enable: bool)
void	set_process_shortcut_input(enable: bool)
void	set_process_unhandled_input(enable: bool)
void	set_process_unhandled_key_input(enable: bool)
void	set_scene_instance_load_placeholder(load_placeholder: bool)
void	set_thread_safe(property: StringName, value: Variant)
void	update_configuration_warnings()

Signals

child_entered_tree(node: Node)

Emitted when the child `node` enters the SceneTree, usually because this node entered the tree (see tree_entered), or add_child has been called.

This signal is emitted *after* the child node's own NOTIFICATION_ENTER_TREE and tree_entered.

child_exiting_tree(node: Node)

Emitted when the child `node` is about to exit the SceneTree, usually because this node is exiting the tree (see tree_exiting), or because the child `node` is being removed or freed.

When this signal is received, the child `node` is still accessible inside the tree. This signal is emitted *after* the child node's own tree_exiting and NOTIFICATION_EXIT_TREE.

child_order_changed()

Emitted when the list of children is changed. This happens when child nodes are added, moved or removed.

editor_description_changed(node: Node)

Emitted when the node's editor description field changed.

ready()

Emitted when the node is considered ready, after _ready is called.

renamed()

Emitted when the node's name is changed, if the node is inside the tree.

replacing_by(node: Node)

Emitted when this node is being replaced by the `node`, see replace_by.

This signal is emitted *after* `node` has been added as a child of the original parent node, but *before* all original child nodes have been reparented to `node`.

---

tree_entered()

Emitted when the node enters the tree.

This signal is emitted *after* the related NOTIFICATION_ENTER_TREE notification.

---

tree_exited()

Emitted after the node exits the tree and is no longer active.

This signal is emitted *after* the related NOTIFICATION_EXIT_TREE notification.

---

tree_exiting()

Emitted when the node is just about to exit the tree. The node is still valid. As such, this is the right place for de-initialization (or a "destructor", if you will).

This signal is emitted *after* the node's _exit_tree, and *before* the related NOTIFICATION_EXIT_TREE.

Enumerations

enum **ProcessMode**:

ProcessMode **PROCESS_MODE_INHERIT** = `0`

Inherits process_mode from the node's parent. This is the default for any newly created node.

ProcessMode **PROCESS_MODE_PAUSABLE** = `1`

Stops processing when SceneTree.paused is `true`. This is the inverse of PROCESS_MODE_WHEN_PAUSED, and the default for the root node.

ProcessMode **PROCESS_MODE_WHEN_PAUSED** = `2`

Process **only** when SceneTree.paused is `true`. This is the inverse of PROCESS_MODE_PAUSABLE.

ProcessMode **PROCESS_MODE_ALWAYS** = `3`

Always process. Keeps processing, ignoring SceneTree.paused. This is the inverse of PROCESS_MODE_DISABLED.

ProcessMode **PROCESS_MODE_DISABLED** = `4`

Never process. Completely disables processing, ignoring SceneTree.paused. This is the inverse of PROCESS_MODE_ALWAYS.

---

enum **ProcessThreadGroup**:

ProcessThreadGroup **PROCESS_THREAD_GROUP_INHERIT** = `0`

Process this node based on the thread group mode of the first parent (or grandparent) node that has a thread group mode that is not inherit. See process_thread_group for more information.

ProcessThreadGroup **PROCESS_THREAD_GROUP_MAIN_THREAD** = `1`

Process this node (and child nodes set to inherit) on the main thread. See process_thread_group for more information.

ProcessThreadGroup **PROCESS_THREAD_GROUP_SUB_THREAD** = `2`

Process this node (and child nodes set to inherit) on a sub-thread. See process_thread_group for more information.

---

flags **ProcessThreadMessages**: 🔗

ProcessThreadMessages **FLAG_PROCESS_THREAD_MESSAGES** = `1`

Allows this node to process threaded messages created with call_deferred_thread_group right before _process is called.

ProcessThreadMessages **FLAG_PROCESS_THREAD_MESSAGES_PHYSICS** = `2`

Allows this node to process threaded messages created with call_deferred_thread_group right before _physics_process is called.

ProcessThreadMessages **FLAG_PROCESS_THREAD_MESSAGES_ALL** = `3`

Allows this node to process threaded messages created with call_deferred_thread_group right before either _process or _physics_process are called.

---

enum **PhysicsInterpolationMode**: 🔗

PhysicsInterpolationMode **PHYSICS_INTERPOLATION_MODE_INHERIT** = `0`

Inherits physics_interpolation_mode from the node's parent. This is the default for any newly created node.

PhysicsInterpolationMode **PHYSICS_INTERPOLATION_MODE_ON** = `1`

Enables physics interpolation for this node and for children set to PHYSICS_INTERPOLATION_MODE_INHERIT. This is the default for the root node.

PhysicsInterpolationMode **PHYSICS_INTERPOLATION_MODE_OFF** = `2`

Disables physics interpolation for this node and for children set to PHYSICS_INTERPOLATION_MODE_INHERIT.

---

enum **DuplicateFlags**:

DuplicateFlags **DUPLICATE_SIGNALS** = `1`

Duplicate the node's signal connections.

DuplicateFlags **DUPLICATE_GROUPS** = `2`

Duplicate the node's groups.

DuplicateFlags **DUPLICATE_SCRIPTS** = `4`

Duplicate the node's script (also overriding the duplicated children's scripts, if combined with DUPLICATE_USE_INSTANTIATION).

DuplicateFlags **DUPLICATE_USE_INSTANTIATION** = `8`

Duplicate using PackedScene.instantiate. If the node comes from a scene saved on disk, re-uses PackedScene.instantiate as the base for the duplicated node and its children.

---

enum **InternalMode**:

InternalMode **INTERNAL_MODE_DISABLED** = `0`

The node will not be internal.

InternalMode **INTERNAL_MODE_FRONT** = `1`

The node will be placed at the beginning of the parent's children, before any non-internal sibling.

InternalMode **INTERNAL_MODE_BACK** = `2`

The node will be placed at the end of the parent's children, after any non-internal sibling.

---

enum **AutoTranslateMode**:

AutoTranslateMode **AUTO_TRANSLATE_MODE_INHERIT** = `0`

Inherits auto_translate_mode from the node's parent. This is the default for any newly created node.

AutoTranslateMode **AUTO_TRANSLATE_MODE_ALWAYS** = `1`

Always automatically translate. This is the inverse of AUTO_TRANSLATE_MODE_DISABLED, and the default for the root node.

AutoTranslateMode **AUTO_TRANSLATE_MODE_DISABLED** = `2`

Never automatically translate. This is the inverse of AUTO_TRANSLATE_MODE_ALWAYS.

String parsing for POT generation will be skipped for this node and children that are set to AUTO_TRANSLATE_MODE_INHERIT.

Constants

NOTIFICATION_ENTER_TREE = `10`

Notification received when the node enters a SceneTree. See _enter_tree.

This notification is received *before* the related tree_entered signal.

NOTIFICATION_EXIT_TREE = `11`

Notification received when the node is about to exit a SceneTree. See _exit_tree.

This notification is received *after* the related tree_exiting signal.

NOTIFICATION_MOVED_IN_PARENT = `12`

Deprecated: This notification is no longer sent by the engine. Use NOTIFICATION_CHILD_ORDER_CHANGED instead.

NOTIFICATION_READY = `13`

Notification received when the node is ready. See _ready.

NOTIFICATION_PAUSED = `14` 🔗

Notification received when the node is paused. See process_mode.

NOTIFICATION_UNPAUSED = `15` 🔗

Notification received when the node is unpaused. See process_mode.

NOTIFICATION_PHYSICS_PROCESS = `16` 🔗

Notification received from the tree every physics frame when is_physics_processing returns `true`. See _physics_process.

NOTIFICATION_PROCESS = `17` 🔗

Notification received from the tree every rendered frame when is_processing returns `true`. See _process.

NOTIFICATION_PARENTED = `18` 🔗

Notification received when the node is set as a child of another node (see add_child and add_sibling).

Note: This does *not* mean that the node entered the SceneTree.

NOTIFICATION_UNPARENTED = `19` 🔗

Notification received when the parent node calls remove_child on this node.

Note: This does *not* mean that the node exited the SceneTree.

NOTIFICATION_SCENE_INSTANTIATED = `20` 🔗

Notification received *only* by the newly instantiated scene root node, when PackedScene.instantiate is completed.

NOTIFICATION_DRAG_BEGIN = `21` 🔗

Notification received when a drag operation begins. All nodes receive this notification, not only the dragged one.

Can be triggered either by dragging a Control that provides drag data (see Control._get_drag_data) or using Control.force_drag.

Use Viewport.gui_get_drag_data to get the dragged data.

NOTIFICATION_DRAG_END = `22`

Notification received when a drag operation ends.

Use Viewport.gui_is_drag_successful to check if the drag succeeded.

NOTIFICATION_PATH_RENAMED = `23`

Notification received when the node's name or one of its ancestors' name is changed. This notification is *not* received when the node is removed from the SceneTree.

NOTIFICATION_CHILD_ORDER_CHANGED = `24`

Notification received when the list of children is changed. This happens when child nodes are added, moved or removed.

NOTIFICATION_INTERNAL_PROCESS = `25`

Notification received from the tree every rendered frame when is_processing_internal returns `true`.

NOTIFICATION_INTERNAL_PHYSICS_PROCESS = `26`

Notification received from the tree every physics frame when is_physics_processing_internal returns `true`.

NOTIFICATION_POST_ENTER_TREE = `27`

Notification received when the node enters the tree, just before NOTIFICATION_READY may be received. Unlike the latter, it is sent every time the node enters tree, not just once.

NOTIFICATION_DISABLED = `28`

Notification received when the node is disabled. See PROCESS_MODE_DISABLED.

NOTIFICATION_ENABLED = `29` 🔗

Notification received when the node is enabled again after being disabled. See PROCESS_MODE_DISABLED.

NOTIFICATION_RESET_PHYSICS_INTERPOLATION = `2001` 🔗

Notification received when reset_physics_interpolation is called on the node or its ancestors.

NOTIFICATION_EDITOR_PRE_SAVE = `9001` 🔗

Notification received right before the scene with the node is saved in the editor. This notification is only sent in the Godot editor and will not occur in exported projects.

NOTIFICATION_EDITOR_POST_SAVE = `9002` 🔗

Notification received right after the scene with the node is saved in the editor. This notification is only sent in the Godot editor and will not occur in exported projects.

NOTIFICATION_WM_MOUSE_ENTER = `1002` 🔗

Notification received when the mouse enters the window.

Implemented for embedded windows and on desktop and web platforms.

NOTIFICATION_WM_MOUSE_EXIT = `1003` 🔗

Notification received when the mouse leaves the window.

Implemented for embedded windows and on desktop and web platforms.

NOTIFICATION_WM_WINDOW_FOCUS_IN = `1004` 🔗

Notification received from the OS when the node's Window ancestor is focused. This may be a change of focus between two windows of the same engine instance, or from the OS desktop or a third-party application to a window of the game (in which case NOTIFICATION_APPLICATION_FOCUS_IN is also received).

A Window node receives this notification when it is focused.

NOTIFICATION_WM_WINDOW_FOCUS_OUT = `1005`

Notification received from the OS when the node's Window ancestor is defocused. This may be a change of focus between two windows of the same engine instance, or from a window of the game to the OS desktop or a third-party application (in which case NOTIFICATION_APPLICATION_FOCUS_OUT is also received).

A Window node receives this notification when it is defocused.

NOTIFICATION_WM_CLOSE_REQUEST = `1006`

Notification received from the OS when a close request is sent (e.g. closing the window with a "Close" button or `Alt + F4`).

Implemented on desktop platforms.

NOTIFICATION_WM_GO_BACK_REQUEST = `1007`

Notification received from the OS when a go back request is sent (e.g. pressing the "Back" button on Android).

Implemented only on Android.

NOTIFICATION_WM_SIZE_CHANGED = `1008`

Notification received when the window is resized.

Note: Only the resized Window node receives this notification, and it's not propagated to the child nodes.

NOTIFICATION_WM_DPI_CHANGE = `1009`

Notification received from the OS when the screen's dots per inch (DPI) scale is changed. Only implemented on macOS.

NOTIFICATION_VP_MOUSE_ENTER = `1010`

Notification received when the mouse cursor enters the Viewport's visible area, that is not occluded behind other Controls or Windows, provided its Viewport.gui_disable_input is `false` and regardless if it's currently focused or not.

NOTIFICATION_VP_MOUSE_EXIT = `1011`

Notification received when the mouse cursor leaves the Viewport's visible area, that is not occluded behind other Controls or Windows, provided its Viewport.gui_disable_input is `false` and regardless if it's currently focused or not.

NOTIFICATION_OS_MEMORY_WARNING = `2009`

Notification received from the OS when the application is exceeding its allocated memory.

Implemented only on iOS.

NOTIFICATION_TRANSLATION_CHANGED = `2010`

Notification received when translations may have changed. Can be triggered by the user changing the locale, changing auto_translate_mode or when the node enters the scene tree. Can be used to respond to language changes, for example to change the UI strings on the fly. Useful when working with the built-in translation support, like Object.tr.

Note: This notification is received alongside NOTIFICATION_ENTER_TREE, so if you are instantiating a scene, the child nodes will not be initialized yet. You can use it to setup translations for this node, child nodes created from script, or if you want to access child nodes added in the editor, make sure the node is ready using is_node_ready.

```
func _notification(what):
    if what == NOTIFICATION_TRANSLATION_CHANGED:
        if not is_node_ready():
            await ready # Wait until ready signal.
        $Label.text = atr("%d Bananas") % banana_counter
```

NOTIFICATION_WM_ABOUT = `2011`

Notification received from the OS when a request for "About" information is sent.

Implemented only on macOS.

NOTIFICATION_CRASH = `2012`

Notification received from Godot's crash handler when the engine is about to crash.

Implemented on desktop platforms, if the crash handler is enabled.

NOTIFICATION_OS_IME_UPDATE = `2013`

Notification received from the OS when an update of the Input Method Engine occurs (e.g. change of IME cursor position or composition string).

Implemented only on macOS.

NOTIFICATION_APPLICATION_RESUMED = `2014`

Notification received from the OS when the application is resumed.

Specific to the Android and iOS platforms.

NOTIFICATION_APPLICATION_PAUSED = `2015`

Notification received from the OS when the application is paused.

Specific to the Android and iOS platforms.

Note: On iOS, you only have approximately 5 seconds to finish a task started by this signal. If you go over this allotment, iOS will kill the app instead of pausing it.

NOTIFICATION_APPLICATION_FOCUS_IN = `2016`

Notification received from the OS when the application is focused, i.e. when changing the focus from the OS desktop or a thirdparty application to any open window of the Godot instance.

Implemented on desktop and mobile platforms.

NOTIFICATION_APPLICATION_FOCUS_OUT = `2017`

Notification received from the OS when the application is defocused, i.e. when changing the focus from any open window of the Godot instance to the OS desktop or a thirdparty application.

Implemented on desktop and mobile platforms.

NOTIFICATION_TEXT_SERVER_CHANGED = `2018`

Notification received when the TextServer is changed.

Property Descriptions

AutoTranslateMode **auto_translate_mode** = `0`

- void **set_auto_translate_mode**(value: AutoTranslateMode)
- AutoTranslateMode **get_auto_translate_mode**()

Defines if any text should automatically change to its translated version depending on the current locale (for nodes such as Label, RichTextLabel, Window, etc.). Also decides if the node's strings should be parsed for POT generation.

Note: For the root node, auto translate mode can also be set via ProjectSettings.internationalization/rendering/root_node_auto_translate.

String **editor_description** = `""`

- void **set_editor_description**(value: String)
- String **get_editor_description**()

An optional description to the node. It will be displayed as a tooltip when hovering over the node in the editor's Scene dock.

MultiplayerAPI **multiplayer** 🔗

- MultiplayerAPI **get_multiplayer**()

The MultiplayerAPI instance associated with this node. See SceneTree.get_multiplayer.

Note: Renaming the node, or moving it in the tree, will not move the MultiplayerAPI to the new path, you will have to update this manually.

StringName **name** 🔗

- void **set_name**(value: StringName)
- StringName **get_name**()

The name of the node. This name must be unique among the siblings (other child nodes from the same parent). When set to an existing sibling's name, the node is automatically renamed.

Note: When changing the name, the following characters will be replaced with an underscore: (`.` `:` `@` `/` `"` `%`). In particular, the `@` character is reserved for auto-generated names. See also String.validate_node_name.

Node **owner** 🔗

- void **set_owner**(value: Node)
- Node **get_owner**()

The owner of this node. The owner must be an ancestor of this node. When packing the owner node in a PackedScene, all the nodes it owns are also saved with it.

Note: In the editor, nodes not owned by the scene root are usually not displayed in the Scene dock, and will **not** be saved. To prevent this, remember to set the owner after calling add_child. See also (see unique_name_in_owner)

PhysicsInterpolationMode **physics_interpolation_mode** = `0` 🔗

- void **set_physics_interpolation_mode**(value: PhysicsInterpolationMode)
- PhysicsInterpolationMode **get_physics_interpolation_mode**()

Allows enabling or disabling physics interpolation per node, offering a finer grain of control than turning physics interpolation on and off globally. See ProjectSettings.physics/common/physics_interpolation and SceneTree.physics_interpolation for the global setting.

Note: When teleporting a node to a distant position you should temporarily disable interpolation with reset_physics_interpolation.

ProcessMode **process_mode** = `0` 🔗

- void **set_process_mode**(value: ProcessMode)
- ProcessMode **get_process_mode**()

The node's processing behavior (see ProcessMode). To check if the node can process in its current mode, use can_process.

int **process_physics_priority** = `0` 🔗

- void **set_physics_process_priority**(value: int)
- int **get_physics_process_priority**()

Similar to process_priority but for NOTIFICATION_PHYSICS_PROCESS, _physics_process or the internal version.

int **process_priority** = `0` 🔗

- void **set_process_priority**(value: int)
- int **get_process_priority**()

The node's execution order of the process callbacks (_process, _physics_process, and internal processing). Nodes whose priority value is *lower* call their process callbacks first, regardless of tree order.

ProcessThreadGroup **process_thread_group** = `0` 🔗

- void **set_process_thread_group**(value: ProcessThreadGroup)
- ProcessThreadGroup **get_process_thread_group**()

Set the process thread group for this node (basically, whether it receives NOTIFICATION_PROCESS, NOTIFICATION_PHYSICS_PROCESS, _process or _physics_process (and the internal versions) on the main thread or in a sub-thread.

By default, the thread group is PROCESS_THREAD_GROUP_INHERIT, which means that this node belongs to the same thread group as the parent node. The thread groups means that nodes in a specific thread group will process together, separate to other thread groups (depending on process_thread_group_order). If the value is set is PROCESS_THREAD_GROUP_SUB_THREAD, this thread group will occur on a sub thread (not the main thread), otherwise if set to PROCESS_THREAD_GROUP_MAIN_THREAD it will process on the main thread. If there is not a parent or grandparent node set to something other than inherit, the node will belong to the *default thread group*. This default group will process on the main thread and its group order is 0.

During processing in a sub-thread, accessing most functions in nodes outside the thread group is forbidden (and it will result in an error in debug mode). Use Object.call_deferred, call_thread_safe, call_deferred_thread_group and the likes in order to communicate from the thread groups to the main thread (or to other thread groups).

To better understand process thread groups, the idea is that any node set to any other value than PROCESS_THREAD_GROUP_INHERIT will include any child (and grandchild) nodes set to inherit into its process thread group. This means that the processing of all the nodes in the group will happen together, at the same time as the node including them.

---

int **process_thread_group_order** 🔗

- void **set_process_thread_group_order**(value: int)
- int **get_process_thread_group_order**()

Change the process thread group order. Groups with a lesser order will process before groups with a greater order. This is useful when a large amount of nodes process in sub thread and, afterwards, another group wants to collect their result in the main thread, as an example.

BitField[ProcessThreadMessages] **process_thread_messages** 🔗

- void **set_process_thread_messages**(value: BitField[ProcessThreadMessages])
- BitField[ProcessThreadMessages] **get_process_thread_messages**()

Set whether the current thread group will process messages (calls to call_deferred_thread_group on threads), and whether it wants to receive them during regular process or physics process callbacks.

String **scene_file_path** 🔗

- void **set_scene_file_path**(value: String)
- String **get_scene_file_path**()

The original scene's file path, if the node has been instantiated from a PackedScene file. Only scene root nodes contains this.

---

bool **unique_name_in_owner** = `false` 🔗

- void **set_unique_name_in_owner**(value: bool)
- bool **is_unique_name_in_owner**()

If `true`, the node can be accessed from any node sharing the same owner or from the owner itself, with special `%Name` syntax in get_node.

Note: If another node with the same owner shares the same name as this node, the other node will no longer be accessible as unique.

Method Descriptions

void **_enter_tree**() virtual

Called when the node enters the SceneTree (e.g. upon instantiating, scene changing, or after calling add_child in a script). If the node has children, its _enter_tree callback will be called first, and then that of the children.

Corresponds to the NOTIFICATION_ENTER_TREE notification in Object._notification.

void **_exit_tree**() virtual

Called when the node is about to leave the SceneTree (e.g. upon freeing, scene changing, or after calling remove_child in a script). If the node has children, its _exit_tree callback will be called last, after all its children have left the tree.

Corresponds to the NOTIFICATION_EXIT_TREE notification in Object._notification and signal tree_exiting. To get notified when the node has already left the active tree, connect to the tree_exited.

PackedStringArray **_get_configuration_warnings**() virtual const

The elements in the array returned from this method are displayed as warnings in the Scene dock if the script that overrides it is a `tool` script.

Returning an empty array produces no warnings.

Call update_configuration_warnings when the warnings need to be updated for this node.

```
@export var energy = 0:
    set(value):
        energy = value
        update_configuration_warnings()

func _get_configuration_warnings():
    if energy < 0:
        return ["Energy must be 0 or greater."]
    else:
        return []
```

void **_input**(event: InputEvent) virtual 🔗

Called when there is an input event. The input event propagates up through the node tree until a node consumes it.

It is only called if input processing is enabled, which is done automatically if this method is overridden, and can be toggled with set_process_input.

To consume the input event and stop it propagating further to other nodes, Viewport.set_input_as_handled can be called.

For gameplay input, _unhandled_input and _unhandled_key_input are usually a better fit as they allow the GUI to intercept the events first.

Note: This method is only called if the node is present in the scene tree (i.e. if it's not an orphan).

---

void **_physics_process**(delta: float) virtual 🔗

Called during the physics processing step of the main loop. Physics processing means that the frame rate is synced to the physics, i.e. the `delta` variable should be constant. `delta` is in seconds.

It is only called if physics processing is enabled, which is done automatically if this method is overridden, and can be toggled with set_physics_process.

Corresponds to the NOTIFICATION_PHYSICS_PROCESS notification in Object._notification.

Note: This method is only called if the node is present in the scene tree (i.e. if it's not an orphan).

---

void **_process**(delta: float) virtual 🔗

Called during the processing step of the main loop. Processing happens at every frame and as fast as possible, so the `delta` time since the previous frame is not constant. `delta` is in seconds.

It is only called if processing is enabled, which is done automatically if this method is overridden, and can be toggled with set_process.

Corresponds to the NOTIFICATION_PROCESS notification in Object._notification.

Note: This method is only called if the node is present in the scene tree (i.e. if it's not an orphan).

---

void **_ready**() virtual

Called when the node is "ready", i.e. when both the node and its children have entered the scene tree. If the node has children, their _ready callbacks get triggered first, and the parent node will receive the ready notification afterwards.

Corresponds to the NOTIFICATION_READY notification in Object._notification. See also the `@onready` annotation for variables.

Usually used for initialization. For even earlier initialization, Object._init may be used. See also _enter_tree.

Note: This method may be called only once for each node. After removing a node from the scene tree and adding it again, _ready will **not** be called a second time. This can be bypassed by requesting another call with request_ready, which may be called anywhere before adding the node again.

---

void **_shortcut_input**(event: InputEvent) virtual

Called when an InputEventKey, InputEventShortcut, or InputEventJoypadButton hasn't been consumed by _input or any GUI Control item. It is called before _unhandled_key_input and _unhandled_input. The input event propagates up through the node tree until a node consumes it.

It is only called if shortcut processing is enabled, which is done automatically if this method is overridden, and can be toggled with set_process_shortcut_input.

To consume the input event and stop it propagating further to other nodes, Viewport.set_input_as_handled can be called.

This method can be used to handle shortcuts. For generic GUI events, use _input instead. Gameplay events should usually be handled with either _unhandled_input or _unhandled_key_input.

Note: This method is only called if the node is present in the scene tree (i.e. if it's not orphan).

void **_unhandled_input**(event: InputEvent) virtual 🔗

Called when an InputEvent hasn't been consumed by _input or any GUI Control item. It is called after _shortcut_input and after _unhandled_key_input. The input event propagates up through the node tree until a node consumes it.

It is only called if unhandled input processing is enabled, which is done automatically if this method is overridden, and can be toggled with set_process_unhandled_input.

To consume the input event and stop it propagating further to other nodes, Viewport.set_input_as_handled can be called.

For gameplay input, this method is usually a better fit than _input, as GUI events need a higher priority. For keyboard shortcuts, consider using _shortcut_input instead, as it is called before this method. Finally, to handle keyboard events, consider using _unhandled_key_input for performance reasons.

Note: This method is only called if the node is present in the scene tree (i.e. if it's not an orphan).

void **_unhandled_key_input**(event: InputEvent) virtual 🔗

Called when an InputEventKey hasn't been consumed by _input or any GUI Control item. It is called after _shortcut_input but before _unhandled_input. The input event propagates up through the node tree until a node consumes it.

It is only called if unhandled key input processing is enabled, which is done automatically if this method is overridden, and can be toggled with set_process_unhandled_key_input.

To consume the input event and stop it propagating further to other nodes, Viewport.set_input_as_handled can be called.

This method can be used to handle Unicode character input with `Alt`, `Alt + Ctrl`, and `Alt + Shift` modifiers, after shortcuts were handled.

For gameplay input, this and _unhandled_input are usually a better fit than _input, as GUI events should be handled first. This method also performs better than _unhandled_input, since unrelated events such as InputEventMouseMotion are automatically filtered. For shortcuts, consider using _shortcut_input instead.

Note: This method is only called if the node is present in the scene tree (i.e. if it's not an orphan).

---

void **add_child**(node: Node, force_readable_name: bool = false, internal: InternalMode = 0) 🔗

Adds a child `node`. Nodes can have any number of children, but every child must have a unique name. Child nodes are automatically deleted when the parent node is deleted, so an entire scene can be removed by deleting its topmost node.

If `force_readable_name` is `true`, improves the readability of the added `node`. If not named, the `node` is renamed to its type, and if it shares name with a sibling, a number is suffixed more appropriately. This operation is very slow. As such, it is recommended leaving this to `false`, which assigns a dummy name featuring `@` in both situations.

If `internal` is different than INTERNAL_MODE_DISABLED, the child will be added as internal node. These nodes are ignored by methods like get_children, unless their parameter `include_internal` is `true`. The intended usage is to hide the internal nodes from the user, so the user won't accidentally delete or modify them. Used by some GUI nodes, e.g. ColorPicker. See InternalMode for available modes.

Note: If `node` already has a parent, this method will fail. Use remove_child first to remove `node` from its current parent. For example:

GDScript

```
var child_node = get_child(0)
if child_node.get_parent():
    child_node.get_parent().remove_child(child_node)
add_child(child_node)
```

C#

```
Node childNode = GetChild(0);
if (childNode.GetParent() != null)
{
    childNode.GetParent().RemoveChild(childNode);
}
AddChild(childNode);
```

If you need the child node to be added below a specific node in the list of children, use add_sibling instead of this method.

Note: If you want a child to be persisted to a PackedScene, you must set owner in addition to calling add_child. This is typically relevant for tool scripts and editor plugins. If add_child is called without setting owner, the newly added **Node** will not be visible in the scene tree, though it will be visible in the 2D/3D view.

---

void **add_sibling**(sibling: Node, force_readable_name: bool = false) 🔗

Adds a `sibling` node to this node's parent, and moves the added sibling right below this node.

If `force_readable_name` is `true`, improves the readability of the added `sibling`. If not named, the `sibling` is renamed to its type, and if it shares name with a sibling, a number is suffixed more appropriately. This operation is very slow. As such, it is recommended leaving this to `false`, which assigns a dummy name featuring `@` in both situations.

Use add_child instead of this method if you don't need the child node to be added below a specific node in the list of children.

Note: If this node is internal, the added sibling will be internal too (see add_child's `internal` parameter).

---

void **add_to_group**(group: StringName, persistent: bool = false) 🔗

Adds the node to the `group`. Groups can be helpful to organize a subset of nodes, for example `"enemies"` or `"collectables"`. See notes in the description, and the group methods in SceneTree.

If `persistent` is `true`, the group will be stored when saved inside a PackedScene. All groups created and displayed in the Node dock are persistent.

Note: To improve performance, the order of group names is *not* guaranteed and may vary between project runs. Therefore, do not rely on the group order.

Note: SceneTree's group methods will *not* work on this node if not inside the tree (see is_inside_tree).

String **atr**(message: String, context: StringName = "") const 🔗

Translates a `message`, using the translation catalogs configured in the Project Settings. Further `context` can be specified to help with the translation. Note that most Control nodes automatically translate their strings, so this method is mostly useful for formatted strings or custom drawn text.

This method works the same as Object.tr, with the addition of respecting the auto_translate_mode state.

If Object.can_translate_messages is `false`, or no translation is available, this method returns the `message` without changes. See Object.set_message_translation.

For detailed examples, see Internationalizing games.

String **atr_n**(message: String, plural_message: StringName, n: int, context: StringName = "") const 🔗

Translates a `message` or `plural_message`, using the translation catalogs configured in the Project Settings. Further `context` can be specified to help with the translation.

This method works the same as Object.tr_n, with the addition of respecting the auto_translate_mode state.

If Object.can_translate_messages is `false`, or no translation is available, this method returns `message` or `plural_message`, without changes. See Object.set_message_translation.

The `n` is the number, or amount, of the message's subject. It is used by the translation system to fetch the correct plural form for the current language.

For detailed examples, see Localization using gettext.

Note: Negative and float numbers may not properly apply to some countable subjects. It's recommended to handle these cases with atr.

Variant **call_deferred_thread_group**(method: StringName, ...) vararg

This function is similar to Object.call_deferred except that the call will take place when the node thread group is processed. If the node thread group processes in sub-threads, then the call will be done on that thread, right before NOTIFICATION_PROCESS or NOTIFICATION_PHYSICS_PROCESS, the _process or _physics_process or their internal versions are called.

Variant **call_thread_safe**(method: StringName, ...) vararg

This function ensures that the calling of this function will succeed, no matter whether it's being done from a thread or not. If called from a thread that is not allowed to call the function, the call will become deferred. Otherwise, the call will go through directly.

---

bool **can_process**() const

Returns `true` if the node can receive processing notifications and input callbacks (NOTIFICATION_PROCESS, _input, etc.) from the SceneTree and

Viewport. The returned value depends on process_mode:

- If set to PROCESS_MODE_PAUSABLE, returns `true` when the game is processing, i.e. SceneTree.paused is `false`;
- If set to PROCESS_MODE_WHEN_PAUSED, returns `true` when the game is paused, i.e. SceneTree.paused is `true`;
- If set to PROCESS_MODE_ALWAYS, always returns `true`;
- If set to PROCESS_MODE_DISABLED, always returns `false`;
- If set to PROCESS_MODE_INHERIT, use the parent node's process_mode to determine the result.

If the node is not inside the tree, returns `false` no matter the value of process_mode.

Tween **create_tween**()

Creates a new Tween and binds it to this node.

This is the equivalent of doing:

GDScript

```
get_tree().create_tween().bind_node(self)
```

C#

```
GetTree().CreateTween().BindNode(this);
```

The Tween will start automatically on the next process frame or physics frame (depending on TweenProcessMode). See Tween.bind_node for more info on Tweens bound to nodes.

Note: The method can still be used when the node is not inside SceneTree. It can fail in an unlikely case of using a custom MainLoop.

Node **duplicate**(flags: int = 15) const

Duplicates the node, returning a new node with all of its properties, signals and groups copied from the original. The behavior can be tweaked through the `flags` (see DuplicateFlags).

Note: For nodes with a Script attached, if Object._init has been defined with required parameters, the duplicated node will not have a Script.

Node **find_child**(pattern: String, recursive: bool = true, owned: bool = true) const 🔗

Finds the first descendant of this node whose name matches `pattern`, returning `null` if no match is found. The matching is done against node names, *not* their paths, through String.match. As such, it is case-sensitive, `"*"` matches zero or more characters, and `"?"` matches any single character.

If `recursive` is `false`, only this node's direct children are checked. Nodes are checked in tree order, so this node's first direct child is checked first, then its own direct children, etc., before moving to the second direct child, and so on. Internal children are also included in the search (see `internal` parameter in add_child).

If `owned` is `true`, only descendants with a valid owner node are checked.

Note: This method can be very slow. Consider storing a reference to the found node in a variable. Alternatively, use get_node with unique names (see unique_name_in_owner).

Note: To find all descendant nodes matching a pattern or a class type, see find_children.

Array[Node] **find_children**(pattern: String, type: String = "", recursive: bool = true, owned: bool = true) const 🔗

Finds all descendants of this node whose names match `pattern`, returning an empty Array if no match is found. The matching is done against node names, *not* their paths, through String.match. As such, it is case-sensitive, `"*"` matches zero or more characters, and `"?"` matches any single character.

If `type` is not empty, only ancestors inheriting from `type` are included (see Object.is_class).

If `recursive` is `false`, only this node's direct children are checked. Nodes are checked in tree order, so this node's first direct child is checked first, then its own direct children, etc., before moving to the second direct child, and so on. Internal children are also included in the search (see `internal` parameter in add_child).

If `owned` is `true`, only descendants with a valid owner node are checked.

Note: This method can be very slow. Consider storing references to the found nodes in a variable.

Note: To find a single descendant node matching a pattern, see find_child.

Node **find_parent**(pattern: String) const 🔗

Finds the first ancestor of this node whose name matches `pattern`, returning `null` if no match is found. The matching is done through String.match. As such, it is case-sensitive, `"*"` matches zero or more characters, and `"?"` matches any single character. See also find_child and find_children.

Note: As this method walks upwards in the scene tree, it can be slow in large, deeply nested nodes. Consider storing a reference to the found node in a variable. Alternatively, use get_node with unique names (see unique_name_in_owner).

Node **get_child**(idx: int, include_internal: bool = false) const 🔗

Fetches a child node by its index. Each child node has an index relative its siblings (see get_index). The first child is at index 0. Negative values can also be used to start from the end of the list. This method can be used in combination with get_child_count to iterate over this node's children. If no child exists at the given index, this method returns `null` and an error is generated.

If `include_internal` is `false`, internal children are ignored (see add_child's `internal` parameter).

```
# Assuming the following are children of this node, in order:
# First, Middle, Last.
```

```
var a = get_child(0).name  # a is "First"
var b = get_child(1).name  # b is "Middle"
var b = get_child(2).name  # b is "Last"
var c = get_child(-1).name # c is "Last"
```

Note: To fetch a node by NodePath, use get_node.

---

int **get_child_count**(include_internal: bool = false) const 🔗

Returns the number of children of this node.

If `include_internal` is `false`, internal children are not counted (see add_child's `internal` parameter).

Array[Node] **get_children**(include_internal: bool = false) const 🔗

Returns all children of this node inside an Array.

If `include_internal` is `false`, excludes internal children from the returned array (see add_child's `internal` parameter).

Array[StringName] **get_groups**() const 🔗

Returns an Array of group names that the node has been added to.

Note: To improve performance, the order of group names is *not* guaranteed and may vary between project runs. Therefore, do not rely on the group order.

Note: This method may also return some group names starting with an underscore (_). These are internally used by the engine. To avoid conflicts, do not use custom groups starting with underscores. To exclude internal groups, see the following code snippet:

GDScript

```
# Stores the node's non-internal groups only (as an array of
StringNames).
```

```
var non_internal_groups = []
for group in get_groups():
    if not str(group).begins_with("_"):
        non_internal_groups.push_back(group)
```

C#

```
// Stores the node's non-internal groups only (as a List of
StringNames).
List<string> nonInternalGroups = new List<string>();
foreach (string group in GetGroups())
{
    if (!group.BeginsWith("_"))
        nonInternalGroups.Add(group);
}
```

int **get_index**(include_internal: bool = false) const

Returns this node's order among its siblings. The first node's index is `0`. See also get_child.

If `include_internal` is `false`, returns the index ignoring internal children. The first, non-internal child will have an index of `0` (see add_child's `internal` parameter).

Window **get_last_exclusive_window**() const

Returns the Window that contains this node, or the last exclusive child in a chain of windows starting with the one that contains this node.

int **get_multiplayer_authority**() const

Returns the peer ID of the multiplayer authority for this node. See set_multiplayer_authority.

Node **get_node**(path: NodePath) const

Fetches a node. The NodePath can either be a relative path (from this node), or an absolute path (from the SceneTree.root) to a node. If `path` does not point to a valid node, generates an error and returns `null`. Attempts to access methods on the return value will result in an *"Attempt to call <method> on a null instance."* error.

Note: Fetching by absolute path only works when the node is inside the scene tree (see is_inside_tree).

Example: Assume this method is called from the Character node, inside the following tree:

```
└─ root
    ├─ Character (you are here!)
    │   ├─ Sword
    │   └─ Backpack
    │       └─ Dagger
    ├─ MyGame
    └─ Swamp
        ├─ Alligator
        ├─ Mosquito
        └─ Goblin
```

The following calls will return a valid node:

GDScript

```
get_node("Sword")
get_node("Backpack/Dagger")
get_node("../Swamp/Alligator")
get_node("/root/MyGame")
```

C#

```
GetNode("Sword");
GetNode("Backpack/Dagger");
GetNode("../Swamp/Alligator");
GetNode("/root/MyGame");
```

Array **get_node_and_resource**(path: NodePath) 🔗

Fetches a node and its most nested resource as specified by the NodePath's subname. Returns an Array of size `3` where:

- Element `0` is the **Node**, or `null` if not found;

- Element `1` is the subname's last nested Resource, or `null` if not found;
- Element `2` is the remaining NodePath, referring to an existing, non-Resource property (see Object.get_indexed).

Example: Assume that the child's Sprite2D.texture has been assigned a AtlasTexture:

GDScript

```
var a = get_node_and_resource("Area2D/Sprite2D")
print(a[0].name) # Prints Sprite2D
print(a[1])      # Prints <null>
print(a[2])      # Prints ^""

var b = get_node_and_resource("Area2D/Sprite2D:texture:atlas")
print(b[0].name)        # Prints Sprite2D
print(b[1].get_class()) # Prints AtlasTexture
print(b[2])             # Prints ^""

var c = get_node_and_resource("Area2D/Sprite2D:texture:atlas:region")
print(c[0].name)        # Prints Sprite2D
print(c[1].get_class()) # Prints AtlasTexture
print(c[2])             # Prints ^":region"
```

C#

```
var a = GetNodeAndResource(NodePath("Area2D/Sprite2D"));
GD.Print(a[0].Name); // Prints Sprite2D
GD.Print(a[1]);      // Prints <null>
GD.Print(a[2]);      // Prints ^"

var b = GetNodeAndResource(NodePath("Area2D/Sprite2D:texture:atlas"));
GD.Print(b[0].name);        // Prints Sprite2D
GD.Print(b[1].get_class()); // Prints AtlasTexture
GD.Print(b[2]);             // Prints ^""

var c =
GetNodeAndResource(NodePath("Area2D/Sprite2D:texture:atlas:region"));
GD.Print(c[0].name);        // Prints Sprite2D
GD.Print(c[1].get_class()); // Prints AtlasTexture
GD.Print(c[2]);             // Prints ^":region"
```

Node **get_node_or_null**(path: NodePath) const 🔗

Fetches a node by NodePath. Similar to get_node, but does not generate an error if `path` does not point to a valid node.

Node **get_parent**() const

Returns this node's parent node, or `null` if the node doesn't have a parent.

NodePath **get_path**() const

Returns the node's absolute path, relative to the SceneTree.root. If the node is not inside the scene tree, this method fails and returns an empty NodePath.

NodePath **get_path_to**(node: Node, use_unique_path: bool = false) const

Returns the relative NodePath from this node to the specified `node`. Both nodes must be in the same SceneTree or scene hierarchy, otherwise this method fails and returns an empty NodePath.

If `use_unique_path` is `true`, returns the shortest path accounting for this node's unique name (see unique_name_in_owner).

Note: If you get a relative path which starts from a unique node, the path may be longer than a normal relative path, due to the addition of the unique node's name.

---

float **get_physics_process_delta_time**() const

Returns the time elapsed (in seconds) since the last physics callback. This value is identical to _physics_process's `delta` parameter, and is often consistent at run-time, unless Engine.physics_ticks_per_second is changed. See also NOTIFICATION_PHYSICS_PROCESS.

---

float **get_process_delta_time**() const

Returns the time elapsed (in seconds) since the last process callback. This value is identical to _process's `delta` parameter, and may vary from frame to frame. See also NOTIFICATION_PROCESS.

bool **get_scene_instance_load_placeholder**() const 🔗

Returns `true` if this node is an instance load placeholder. See InstancePlaceholder and set_scene_instance_load_placeholder.

SceneTree **get_tree**() const 🔗

Returns the SceneTree that contains this node. If this node is not inside the tree, generates an error and returns `null`. See also is_inside_tree.

String **get_tree_string**() 🔗

Returns the tree as a String. Used mainly for debugging purposes. This version displays the path relative to the current node, and is good for copy/pasting into the get_node function. It also can be used in game UI/UX.

May print, for example:

```
TheGame
TheGame/Menu
TheGame/Menu/Label
TheGame/Menu/Camera2D
TheGame/SplashScreen
TheGame/SplashScreen/Camera2D
```

String **get_tree_string_pretty**() 🔗

Similar to get_tree_string, this returns the tree as a String. This version displays a more graphical representation similar to what is displayed in the Scene Dock. It is useful for inspecting larger trees.

May print, for example:

```
└─ TheGame
   ├─ Menu
   |   ├─ Label
   |   └─ Camera2D
   └─ SplashScreen
       └─ Camera2D
```

Viewport **get_viewport**() const

Returns the node's closest Viewport ancestor, if the node is inside the tree. Otherwise, returns `null`.

Window **get_window**() const

Returns the Window that contains this node. If the node is in the main window, this is equivalent to getting the root node (`get_tree().get_root()`).

bool **has_node**(path: NodePath) const

Returns `true` if the `path` points to a valid node. See also get_node.

bool **has_node_and_resource**(path: NodePath) const

Returns `true` if `path` points to a valid node and its subnames point to a valid Resource, e.g. `Area2D/CollisionShape2D:shape`. Properties that are not Resource types (such as nodes or other Variant types) are not considered. See also get_node_and_resource.

bool **is_ancestor_of**(node: Node) const

Returns `true` if the given `node` is a direct or indirect child of this node.

bool **is_displayed_folded**() const

Returns `true` if the node is folded (collapsed) in the Scene dock. This method is intended to be used in editor plugins and tools. See also set_display_folded.

---

bool **is_editable_instance**(node: Node) const

Returns `true` if `node` has editable children enabled relative to this node. This method is intended to be used in editor plugins and tools. See also set_editable_instance.

---

bool **is_greater_than**(node: Node) const

Returns `true` if the given `node` occurs later in the scene hierarchy than this node. A node occurring later is usually processed last.

---

bool **is_in_group**(group: StringName) const

Returns `true` if this node has been added to the given `group`. See add_to_group and remove_from_group. See also notes in the description, and the SceneTree's group methods.

---

bool **is_inside_tree**() const

Returns `true` if this node is currently inside a SceneTree. See also get_tree.

---

bool **is_multiplayer_authority**() const

Returns `true` if the local system is the multiplayer authority of this node.

---

bool **is_node_ready**() const

Returns `true` if the node is ready, i.e. it's inside scene tree and all its children are initialized.

request_ready resets it back to `false`.

---

bool **is_part_of_edited_scene**() const

Returns `true` if the node is part of the scene currently opened in the editor.

---

bool **is_physics_interpolated**() const

Returns `true` if physics interpolation is enabled for this node (see physics_interpolation_mode).

Note: Interpolation will only be active if both the flag is set **and** physics interpolation is enabled within the SceneTree. This can be tested using is_physics_interpolated_and_enabled.

---

bool **is_physics_interpolated_and_enabled**() const

Returns `true` if physics interpolation is enabled (see physics_interpolation_mode) **and** enabled in the SceneTree.

This is a convenience version of is_physics_interpolated that also checks whether physics interpolation is enabled globally.

See SceneTree.physics_interpolation and ProjectSettings.physics/common/physics_interpolation.

---

bool **is_physics_processing**() const

Returns `true` if physics processing is enabled (see set_physics_process).

---

bool **is_physics_processing_internal**() const

Returns `true` if internal physics processing is enabled (see set_physics_process_internal).

---

bool **is_processing**() const

Returns `true` if processing is enabled (see set_process).

---

bool **is_processing_input**() const 🔗

Returns `true` if the node is processing input (see set_process_input).

---

bool **is_processing_internal**() const 🔗

Returns `true` if internal processing is enabled (see set_process_internal).

---

bool **is_processing_shortcut_input**() const 🔗

Returns `true` if the node is processing shortcuts (see set_process_shortcut_input).

---

bool **is_processing_unhandled_input**() const 🔗

Returns `true` if the node is processing unhandled input (see set_process_unhandled_input).

---

bool **is_processing_unhandled_key_input**() const 🔗

Returns `true` if the node is processing unhandled key input (see set_process_unhandled_key_input).

---

void **move_child**(child_node: Node, to_index: int) 🔗

Moves `child_node` to the given index. A node's index is the order among its siblings. If `to_index` is negative, the index is counted from the end of the list. See also get_child and get_index.

Note: The processing order of several engine callbacks (_ready, _process, etc.) and notifications sent through propagate_notification is affected by tree order. CanvasItem nodes are also rendered in tree order. See also process_priority.

---

void **notify_deferred_thread_group**(what: int) 🔗

Similar to call_deferred_thread_group, but for notifications.

void **notify_thread_safe**(what: int) 🔗

Similar to call_thread_safe, but for notifications.

void **print_orphan_nodes**() static 🔗

Prints all orphan nodes (nodes outside the SceneTree). Useful for debugging.

Note: This method only works in debug builds. Does nothing in a project exported in release mode.

void **print_tree**() 🔗

Prints the node and its children to the console, recursively. The node does not have to be inside the tree. This method outputs NodePaths relative to this node, and is good for copy/pasting into get_node. See also print_tree_pretty.

May print, for example:

```
.
Menu
Menu/Label
Menu/Camera2D
SplashScreen
SplashScreen/Camera2D
```

void **print_tree_pretty**() 🔗

Prints the node and its children to the console, recursively. The node does not have to be inside the tree. Similar to print_tree, but the graphical representation looks like what is displayed in the editor's Scene dock. It is useful for inspecting larger trees.

May print, for example:

```
┖╴TheGame
   ┠╴Menu
   ┃  ┠╴Label
   ┃  ┖╴Camera2D
   ┖╴SplashScreen
      ┖╴Camera2D
```

---

void **propagate_call**(method: StringName, args: Array = [], parent_first: bool = false) 🔗

Calls the given `method` name, passing `args` as arguments, on this node and all of its children, recursively.

If `parent_first` is `true`, the method is called on this node first, then on all of its children. If `false`, the children's methods are called first.

---

void **propagate_notification**(what: int) 🔗

Calls Object.notification with `what` on this node and all of its children, recursively.

---

void **queue_free**() 🔗

Queues this node to be deleted at the end of the current frame. When deleted, all of its children are deleted as well, and all references to the node and its children become invalid.

Unlike with Object.free, the node is not deleted instantly, and it can still be accessed before deletion. It is also safe to call queue_free multiple times. Use Object.is_queued_for_deletion to check if the node will be deleted at the end of the frame.

Note: The node will only be freed after all other deferred calls are finished. Using this method is not always the same as calling Object.free through Object.call_deferred.

---

void **remove_child**(node: Node)

Removes a child `node`. The `node`, along with its children, are **not** deleted. To delete a node, see queue_free.

Note: When this node is inside the tree, this method sets the owner of the removed `node` (or its descendants) to `null`, if their owner is no longer an ancestor (see is_ancestor_of).

---

void **remove_from_group**(group: StringName)

Removes the node from the given `group`. Does nothing if the node is not in the `group`. See also notes in the description, and the SceneTree's group methods.

---

void **reparent**(new_parent: Node, keep_global_transform: bool = true)

Changes the parent of this **Node** to the `new_parent`. The node needs to already have a parent. The node's owner is preserved if its owner is still reachable from the new location (i.e., the node is still a descendant of the new parent after the operation).

If `keep_global_transform` is `true`, the node's global transform will be preserved if supported. Node2D, Node3D and Control support this argument (but Control keeps only position).

---

void **replace_by**(node: Node, keep_groups: bool = false)

Replaces this node by the given `node`. All children of this node are moved to `node`.

If `keep_groups` is `true`, the `node` is added to the same groups that the replaced node is in (see add_to_group).

Warning: The replaced node is removed from the tree, but it is **not** deleted. To prevent memory leaks, store a reference to the node in a variable, or use Object.free.

---

void **request_ready**() 🔗

Requests _ready to be called again the next time the node enters the tree. Does **not** immediately call _ready.

Note: This method only affects the current node. If the node's children also need to request ready, this method needs to be called for each one of them. When the node and its children enter the tree again, the order of _ready callbacks will be the same as normal.

---

void **reset_physics_interpolation**() 🔗

When physics interpolation is active, moving a node to a radically different transform (such as placement within a level) can result in a visible glitch as the object is rendered moving from the old to new position over the physics tick.

That glitch can be prevented by calling this method, which temporarily disables interpolation until the physics tick is complete.

The notification NOTIFICATION_RESET_PHYSICS_INTERPOLATION will be received by the node and all children recursively.

Note: This function should be called **after** moving the node, rather than before.

Error **rpc**(method: StringName, ...) vararg 🔗

Sends a remote procedure call request for the given `method` to peers on the network (and locally), sending additional arguments to the method called by the RPC. The call request will only be received by nodes with the same NodePath, including the exact same name. Behavior depends on the RPC configuration for the given `method` (see rpc_config and @GDScript.@rpc). By default, methods are not exposed to RPCs.

May return @GlobalScope.OK if the call is successful, @GlobalScope.ERR_INVALID_PARAMETER if the arguments passed in the `method` do not match, @GlobalScope.ERR_UNCONFIGURED if the node's multiplayer cannot be fetched (such as when the node is not inside the tree), @GlobalScope.ERR_CONNECTION_ERROR if multiplayer's connection is not available.

Note: You can only safely use RPCs on clients after you received the MultiplayerAPI.connected_to_server signal from the MultiplayerAPI. You also need to keep track of the connection state, either by the MultiplayerAPI signals like MultiplayerAPI.server_disconnected or by checking (`get_multiplayer().peer.get_connection_status() == CONNECTION_CONNECTED`).

---

void **rpc_config**(method: StringName, config: Variant) 🔗

Changes the RPC configuration for the given `method`. `config` should either be `null` to disable the feature (as by default), or a Dictionary containing the following entries:

- `rpc_mode`: see RPCMode;
- `transfer_mode`: see TransferMode;
- `call_local`: if `true`, the method will also be called locally;
- `channel`: an int representing the channel to send the RPC on.

Note: In GDScript, this method corresponds to the @GDScript.@rpc annotation, with various parameters passed (`@rpc(any)`, `@rpc(authority)`...). See also the high-level multiplayer tutorial.

Error **rpc_id**(peer_id: int, method: StringName, ...) vararg 🔗

Sends a rpc to a specific peer identified by `peer_id` (see MultiplayerPeer.set_target_peer).

May return @GlobalScope.OK if the call is successful, @GlobalScope.ERR_INVALID_PARAMETER if the arguments passed in the `method` do not match, @GlobalScope.ERR_UNCONFIGURED if the node's multiplayer cannot be fetched (such as when the node is not inside the tree), @GlobalScope.ERR_CONNECTION_ERROR if multiplayer's connection is not available.

---

void **set_deferred_thread_group**(property: StringName, value: Variant)

Similar to call_deferred_thread_group, but for setting properties.

void **set_display_folded**(fold: bool)

If set to `true`, the node appears folded in the Scene dock. As a result, all of its children are hidden. This method is intended to be used in editor plugins and tools, but it also works in release builds. See also is_displayed_folded.

void **set_editable_instance**(node: Node, is_editable: bool)

Set to `true` to allow all nodes owned by `node` to be available, and editable, in the Scene dock, even if their owner is not the scene root. This method is intended to be used in editor plugins and tools, but it also works in release builds. See also is_editable_instance.

void **set_multiplayer_authority**(id: int, recursive: bool = true)

Sets the node's multiplayer authority to the peer with the given peer `id`. The multiplayer authority is the peer that has authority over the node on the network. Defaults to peer ID 1 (the server). Useful in conjunction with rpc_config and the MultiplayerAPI.

If `recursive` is `true`, the given peer is recursively set as the authority for all children of this node.

Warning: This does **not** automatically replicate the new authority to other peers. It is the developer's responsibility to do so. You may replicate the new authority's information using MultiplayerSpawner.spawn_function, an RPC, or a MultiplayerSynchronizer. Furthermore, the parent's authority does **not** propagate to newly added children.

void **set_physics_process**(enable: bool)

If set to `true`, enables physics (fixed framerate) processing. When a node is being processed, it will receive a NOTIFICATION_PHYSICS_PROCESS at a fixed (usually 60 FPS, see Engine.physics_ticks_per_second to change) interval (and the _physics_process callback will be called if it exists).

Note: If _physics_process is overridden, this will be automatically enabled before _ready is called.

---

void **set_physics_process_internal**(enable: bool)

If set to `true`, enables internal physics for this node. Internal physics processing happens in isolation from the normal _physics_process calls and is used by some nodes internally to guarantee proper functioning even if the node is paused or physics processing is disabled for scripting (set_physics_process).

Warning: Built-in nodes rely on internal processing for their internal logic. Disabling it is unsafe and may lead to unexpected behavior. Use this method if you know what you are doing.

---

void **set_process**(enable: bool)

If set to `true`, enables processing. When a node is being processed, it will receive a NOTIFICATION_PROCESS on every drawn frame (and the _process callback will be called if it exists).

Note: If _process is overridden, this will be automatically enabled before _ready is called.

Note: This method only affects the _process callback, i.e. it has no effect on other callbacks like _physics_process. If you want to disable all processing for the node, set process_mode to PROCESS_MODE_DISABLED.

---

void **set_process_input**(enable: bool)

If set to `true`, enables input processing.

Note: If _input is overridden, this will be automatically enabled before _ready is called. Input processing is also already enabled for GUI controls, such as Button and TextEdit.

void **set_process_internal**(enable: bool)

If set to `true`, enables internal processing for this node. Internal processing happens in isolation from the normal _process calls and is used by some nodes internally to guarantee proper functioning even if the node is paused or processing is disabled for scripting (set_process).

Warning: Built-in nodes rely on internal processing for their internal logic. Disabling it is unsafe and may lead to unexpected behavior. Use this method if you know what you are doing.

void **set_process_shortcut_input**(enable: bool)

If set to `true`, enables shortcut processing for this node.

Note: If _shortcut_input is overridden, this will be automatically enabled before _ready is called.

void **set_process_unhandled_input**(enable: bool)

If set to `true`, enables unhandled input processing. It enables the node to receive all input that was not previously handled (usually by a Control).

Note: If _unhandled_input is overridden, this will be automatically enabled before _ready is called. Unhandled input processing is also already enabled for GUI controls, such as Button and TextEdit.

void **set_process_unhandled_key_input**(enable: bool)

If set to `true`, enables unhandled key input processing.

Note: If _unhandled_key_input is overridden, this will be automatically enabled before _ready is called.

---

void **set_scene_instance_load_placeholder**(load_placeholder: bool) 🔗

If set to `true`, the node becomes a InstancePlaceholder when packed and instantiated from a PackedScene. See also get_scene_instance_load_placeholder.

---

void **set_thread_safe**(property: StringName, value: Variant) 🔗

Similar to call_thread_safe, but for setting properties.

---

void **update_configuration_warnings**() 🔗

Refreshes the warnings displayed for this node in the Scene dock. Use _get_configuration_warnings to customize the warning messages to display.

AcceptDialog

Inherits: Window **<** Viewport **<** Node **<** Object

Inherited By: ConfirmationDialog

A base dialog used for user notification.

Description

The default use of **AcceptDialog** is to allow it to only be accepted or closed, with the same result. However, the confirmed and canceled signals allow to make the two actions different, and the add_button method allows to add custom buttons and actions.

Properties

bool	dialog_autowrap	`false`
bool	dialog_close_on_escape	`true`
bool	dialog_hide_on_ok	`true`
String	dialog_text	`""`
bool	exclusive	`true` (overrides Window)

bool	keep_title_visible	`true` (overrides Window)
String	ok_button_text	`"OK"`
String	title	`"Alert!"` (overrides Window)
bool	transient	`true` (overrides Window)
bool	visible	`false` (overrides Window)
bool	wrap_controls	`true` (overrides Window)

Methods

Button	add_button(text: String, right: bool = false, action: String = "")
Button	add_cancel_button(name: String)
Label	get_label()
Button	get_ok_button()
void	register_text_enter(line_edit: LineEdit)

void	remove_button(button: Button)

Theme Properties

int	buttons_min_height	`0`
int	buttons_min_width	`0`
int	buttons_separation	`10`
StyleBox	panel	

Signals

canceled() 🔗

Emitted when the dialog is closed or the button created with add_cancel_button is pressed.

confirmed() 🔗

Emitted when the dialog is accepted, i.e. the OK button is pressed.

custom_action(action: StringName) 🔗

Emitted when a custom button is pressed. See add_button.

Property Descriptions

bool **dialog_autowrap** = `false`

- void **set_autowrap**(value: bool)
- bool **has_autowrap**()

Sets autowrapping for the text in the dialog.

---

bool **dialog_close_on_escape** = `true`

- void **set_close_on_escape**(value: bool)
- bool **get_close_on_escape**()

If `true`, the dialog will be hidden when the escape key (@GlobalScope.KEY_ESCAPE) is pressed.

---

bool **dialog_hide_on_ok** = `true`

- void **set_hide_on_ok**(value: bool)
- bool **get_hide_on_ok**()

If `true`, the dialog is hidden when the OK button is pressed. You can set it to `false` if you want to do e.g. input validation when receiving the confirmed signal, and handle hiding the dialog in your own logic.

Note: Some nodes derived from this class can have a different default value, and potentially their own built-in logic overriding this setting. For example FileDialog defaults to `false`, and has its own input validation code that is called when you press OK, which eventually hides the dialog if the input is valid. As such, this property can't be used in FileDialog to disable hiding the dialog when pressing OK.

String **dialog_text** = `""` 🔗

- void **set_text**(value: String)
- String **get_text**()

The text displayed by the dialog.

String **ok_button_text** = `"OK"` 🔗

- void **set_ok_button_text**(value: String)
- String **get_ok_button_text**()

The text displayed by the OK button (see get_ok_button).

Method Descriptions

Button **add_button**(text: String, right: bool = false, action: String = "") 🔗

Adds a button with label `text` and a custom `action` to the dialog and returns the created button. `action` will be passed to the custom_action signal when pressed.

If `true`, `right` will place the button to the right of any sibling buttons.

You can use remove_button method to remove a button created with this method from the dialog.

Button **add_cancel_button**(name: String) 🔗

Adds a button with label `name` and a cancel action to the dialog and returns the created button.

You can use remove_button method to remove a button created with this method from the dialog.

Label **get_label**() 🔗

Returns the label used for built-in text.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their CanvasItem.visible property.

Button **get_ok_button**() 🔗

Returns the OK Button instance.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their CanvasItem.visible property.

---

void **register_text_enter**(line_edit: LineEdit) 🔗

Registers a LineEdit in the dialog. When the enter key is pressed, the dialog will be accepted.

---

void **remove_button**(button: Button) 🔗

Removes the `button` from the dialog. Does NOT free the `button`. The `button` must be a Button added with add_button or add_cancel_button method. After removal, pressing the `button` will no longer emit this dialog's custom_action or canceled signals.

Theme Property Descriptions

int **buttons_min_height** = `0`

The minimum height of each button in the bottom row (such as OK/Cancel) in pixels. This can be increased to make buttons with short texts easier to click/tap.

int **buttons_min_width** = `0`

The minimum width of each button in the bottom row (such as OK/Cancel) in pixels. This can be increased to make buttons with short texts easier to click/tap.

int **buttons_separation** = `10`

The size of the vertical space between the dialog's content and the button row.

StyleBox **panel**

The panel that fills the background of the window.

AnimatableBody2D

Inherits: StaticBody2D **<** PhysicsBody2D **<** CollisionObject2D **<** Node2D **<** CanvasItem **<** Node **<** Object

A 2D physics body that can't be moved by external forces. When moved manually, it affects other bodies in its path.

Description

An animatable 2D physics body. It can't be moved by external forces or contacts, but can be moved manually by other means such as code, AnimationMixers (with AnimationMixer.callback_mode_process set to AnimationMixer.ANIMATION_CALLBACK_MODE_PROCESS_PHYSICS), and RemoteTransform2D.

When **AnimatableBody2D** is moved, its linear and angular velocity are estimated and used to affect other physics bodies in its path. This makes it useful for moving platforms, doors, and other moving objects.

Properties

bool	sync_to_physics	`true`

Property Descriptions

bool **sync_to_physics** = `true` 🔗

- void **set_sync_to_physics**(value: bool)
- bool **is_sync_to_physics_enabled**()

If `true`, the body's movement will be synchronized to the physics frame. This is useful when animating movement via AnimationPlayer, for example on moving platforms. Do **not** use together with PhysicsBody2D.move_and_collide.

AnimatableBody3D

Inherits: StaticBody3D **<** PhysicsBody3D **<** CollisionObject3D **<** Node3D **<** Node **<** Object

A 3D physics body that can't be moved by external forces. When moved manually, it affects other bodies in its path.

Description

An animatable 3D physics body. It can't be moved by external forces or contacts, but can be moved manually by other means such as code, AnimationMixers (with AnimationMixer.callback_mode_process set to AnimationMixer.ANIMATION_CALLBACK_MODE_PROCESS_PHYSICS), and RemoteTransform3D.

When **AnimatableBody3D** is moved, its linear and angular velocity are estimated and used to affect other physics bodies in its path. This makes it useful for moving platforms, doors, and other moving objects.

Tutorials

- 3D Physics Tests Demo [https://godotengine.org/asset-library/asset/2747]
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]
- 3D Voxel Demo [https://godotengine.org/asset-library/asset/2755]

Properties

bool	sync_to_physics	`true`

Property Descriptions

bool **sync_to_physics** = `true` 🔗

- void **set_sync_to_physics**(value: bool)
- bool **is_sync_to_physics_enabled**()

If `true`, the body's movement will be synchronized to the physics frame. This is useful when animating movement via AnimationPlayer, for example on moving platforms. Do **not** use together with PhysicsBody3D.move_and_collide.

AnimatedSprite2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Sprite node that contains multiple textures as frames to play for animation.

Description

AnimatedSprite2D is similar to the Sprite2D node, except it carries multiple textures as animation frames. Animations are created using a SpriteFrames resource, which allows you to import image files (or a folder containing said files) to provide the animation frames for the sprite. The SpriteFrames resource can be configured in the editor via the SpriteFrames bottom panel.

Tutorials

- 2D Sprite animation
- 2D Dodge The Creeps Demo [https://godotengine.org/asset-library/asset/2712]

Properties

StringName	animation	`&"default"`
String	autoplay	`""`

bool	centered	`true`
bool	flip_h	`false`
bool	flip_v	`false`
int	frame	`0`
float	frame_progress	`0.0`
Vector2	offset	`Vector2(0, 0)`
float	speed_scale	`1.0`
SpriteFrames	sprite_frames	

Methods

float	get_playing_speed() const
bool	is_playing() const
void	pause()

void	play(name: StringName = &"", custom_speed: float = 1.0, from_end: bool = false)
void	play_backwards(name: StringName = &"")
void	set_frame_and_progress(frame: int, progress: float)
void	stop()

Signals

animation_changed()

Emitted when animation changes.

---

animation_finished()

Emitted when the animation reaches the end, or the start if it is played in reverse. When the animation finishes, it pauses the playback.

Note: This signal is not emitted if an animation is looping.

---

animation_looped()

Emitted when the animation loops.

frame_changed()

Emitted when frame changes.

---

sprite_frames_changed()

Emitted when sprite_frames changes.

Property Descriptions

StringName **animation** = `&"default"`

- void **set_animation**(value: StringName)
- StringName **get_animation**()

The current animation from the sprite_frames resource. If this value is changed, the frame counter and the frame_progress are reset.

String **autoplay** = `""`

- void **set_autoplay**(value: String)
- String **get_autoplay**()

The key of the animation to play when the scene loads.

---

bool **centered** = `true`

- void **set_centered**(value: bool)
- bool **is_centered**()

If `true`, texture will be centered.

Note: For games with a pixel art aesthetic, textures may appear deformed when centered. This is caused by their position being between pixels. To prevent this, set this property to `false`, or consider enabling ProjectSettings.rendering/2d/snap/snap_2d_vertices_to_pixel and ProjectSettings.rendering/2d/snap/snap_2d_transforms_to_pixel.

---

bool **flip_h** = `false` 🔗

- void **set_flip_h**(value: bool)
- bool **is_flipped_h**()

If `true`, texture is flipped horizontally.

---

bool **flip_v** = `false` 🔗

- void **set_flip_v**(value: bool)
- bool **is_flipped_v**()

If `true`, texture is flipped vertically.

---

int **frame** = `0` 🔗

- void **set_frame**(value: int)
- int **get_frame**()

The displayed animation frame's index. Setting this property also resets frame_progress. If this is not desired, use set_frame_and_progress.

---

float **frame_progress** = `0.0` 🔗

- void **set_frame_progress**(value: float)

- float **get_frame_progress**()

The progress value between `0.0` and `1.0` until the current frame transitions to the next frame. If the animation is playing backwards, the value transitions from `1.0` to `0.0`.

Vector2 **offset** = `Vector2(0, 0)` 🔗

- void **set_offset**(value: Vector2)
- Vector2 **get_offset**()

The texture's drawing offset.

float **speed_scale** = `1.0` 🔗

- void **set_speed_scale**(value: float)
- float **get_speed_scale**()

The speed scaling ratio. For example, if this value is `1`, then the animation plays at normal speed. If it's `0.5`, then it plays at half speed. If it's `2`, then it plays at double speed.

If set to a negative value, the animation is played in reverse. If set to `0`, the animation will not advance.

SpriteFrames **sprite_frames** 🔗

- void **set_sprite_frames**(value: SpriteFrames)
- SpriteFrames **get_sprite_frames**()

The SpriteFrames resource containing the animation(s). Allows you the option to load, edit, clear, make unique and save the states of the SpriteFrames resource.

Method Descriptions

float **get_playing_speed**() const

Returns the actual playing speed of current animation or `0` if not playing. This speed is the speed_scale property multiplied by `custom_speed` argument specified when calling the play method.

Returns a negative value if the current animation is playing backwards.

---

bool **is_playing**() const

Returns `true` if an animation is currently playing (even if speed_scale and/or `custom_speed` are `0`).

---

void **pause**()

Pauses the currently playing animation. The frame and frame_progress will be kept and calling play or play_backwards without arguments will resume the animation from the current playback position.

See also stop.

---

void **play**(name: StringName = &"", custom_speed: float = 1.0, from_end: bool = false)

Plays the animation with key `name`. If `custom_speed` is negative and `from_end` is `true`, the animation will play backwards (which is equivalent to calling play_backwards).

If this method is called with that same animation `name`, or with no `name` parameter, the assigned animation will resume playing if it was paused.

void **play_backwards**(name: StringName = &"")

Plays the animation with key `name` in reverse.

This method is a shorthand for play with `custom_speed = -1.0` and `from_end = true`, so see its description for more information.

void **set_frame_and_progress**(frame: int, progress: float)

The setter of frame resets the frame_progress to `0.0` implicitly, but this method avoids that.

This is useful when you want to carry over the current frame_progress to another frame.

Example:

GDScript

```gdscript
# Change the animation with keeping the frame index and progress.
var current_frame = animated_sprite.get_frame()
var current_progress = animated_sprite.get_frame_progress()
animated_sprite.play("walk_another_skin")
animated_sprite.set_frame_and_progress(current_frame, current_progress)
```

void **stop**()

Stops the currently playing animation. The animation position is reset to `0` and the `custom_speed` is reset to `1.0`. See also pause.

AnimatedSprite3D

Inherits: SpriteBase3D **<** GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

2D sprite node in 3D world, that can use multiple 2D textures for animation.

Description

AnimatedSprite3D is similar to the Sprite3D node, except it carries multiple textures as animation sprite_frames. Animations are created using a SpriteFrames resource, which allows you to import image files (or a folder containing said files) to provide the animation frames for the sprite. The SpriteFrames resource can be configured in the editor via the SpriteFrames bottom panel.

Tutorials

- 2D Sprite animation (also applies to 3D)

Properties

StringName	animation	`&"default"`
String	autoplay	`""`
int	frame	`0`

float	frame_progress	`0.0`
float	speed_scale	`1.0`
SpriteFrames	sprite_frames	

Methods

float	get_playing_speed() const
bool	is_playing() const
void	pause()
void	play(name: StringName = &"", custom_speed: float = 1.0, from_end: bool = false)
void	play_backwards(name: StringName = &"")
void	set_frame_and_progress(frame: int, progress: float)
void	stop()

Signals

animation_changed() 🔗

Emitted when animation changes.

---

animation_finished() 🔗

Emitted when the animation reaches the end, or the start if it is played in reverse. When the animation finishes, it pauses the playback.

Note: This signal is not emitted if an animation is looping.

---

animation_looped() 🔗

Emitted when the animation loops.

---

frame_changed() 🔗

Emitted when frame changes.

---

sprite_frames_changed() 🔗

Emitted when sprite_frames changes.

Property Descriptions

StringName **animation** = `&"default"` 🔗

- void **set_animation**(value: StringName)
- StringName **get_animation**()

The current animation from the sprite_frames resource. If this value is changed, the frame counter and the frame_progress are reset.

String **autoplay** = `""` 🔗

- void **set_autoplay**(value: String)
- String **get_autoplay**()

The key of the animation to play when the scene loads.

int **frame** = `0` 🔗

- void **set_frame**(value: int)
- int **get_frame**()

The displayed animation frame's index. Setting this property also resets frame_progress. If this is not desired, use set_frame_and_progress.

float **frame_progress** = `0.0` 🔗

- void **set_frame_progress**(value: float)
- float **get_frame_progress**()

The progress value between `0.0` and `1.0` until the current frame transitions to the next frame. If the animation is playing backwards, the value transitions from `1.0` to `0.0`.

float **speed_scale** = `1.0` 🔗

- void **set_speed_scale**(value: float)
- float **get_speed_scale**()

The speed scaling ratio. For example, if this value is `1`, then the animation plays at normal speed. If it's `0.5`, then it plays at half speed. If it's `2`, then it plays at double speed.

If set to a negative value, the animation is played in reverse. If set to `0`, the animation will not advance.

SpriteFrames **sprite_frames**

- void **set_sprite_frames**(value: SpriteFrames)
- SpriteFrames **get_sprite_frames**()

The SpriteFrames resource containing the animation(s). Allows you the option to load, edit, clear, make unique and save the states of the SpriteFrames resource.

Method Descriptions

float **get_playing_speed**() const

Returns the actual playing speed of current animation or `0` if not playing. This speed is the speed_scale property multiplied by `custom_speed` argument specified when calling the play method.

Returns a negative value if the current animation is playing backwards.

---

bool **is_playing**() const

Returns `true` if an animation is currently playing (even if speed_scale and/or `custom_speed` are `0`).

---

void **pause**()

Pauses the currently playing animation. The frame and frame_progress will be kept and calling play or play_backwards without arguments will resume the animation from the current playback position.

See also stop.

---

void **play**(name: StringName = &"", custom_speed: float = 1.0, from_end: bool = false)

Plays the animation with key `name`. If `custom_speed` is negative and `from_end` is `true`, the animation will play backwards (which is equivalent to calling play_backwards).

If this method is called with that same animation `name`, or with no `name` parameter, the assigned animation will resume playing if it was paused.

---

void **play_backwards**(name: StringName = &"")

Plays the animation with key `name` in reverse.

This method is a shorthand for play with `custom_speed = -1.0` and `from_end = true`, so see its description for more information.

---

void **set_frame_and_progress**(frame: int, progress: float)

The setter of frame resets the frame_progress to `0.0` implicitly, but this method avoids that.

This is useful when you want to carry over the current frame_progress to another frame.

Example:

GDScript

```
# Change the animation with keeping the frame index and progress.
var current_frame = animated_sprite.get_frame()
var current_progress = animated_sprite.get_frame_progress()
animated_sprite.play("walk_another_skin")
animated_sprite.set_frame_and_progress(current_frame, current_progress)
```

---

void **stop**()

Stops the currently playing animation. The animation position is reset to `0` and the `custom_speed` is reset to `1.0`. See also pause.

AnimationMixer

Inherits: Node < Object

Inherited By: AnimationPlayer, AnimationTree

Base class for AnimationPlayer and AnimationTree.

Description

Base class for AnimationPlayer and AnimationTree to manage animation lists. It also has general properties and methods for playback and blending.

After instantiating the playback information data within the extended class, the blending is processed by the **AnimationMixer**.

Tutorials

- Migrating Animations from Godot 4.0 to 4.3 [https://godotengine.org/article/migrating-animations-from-godot-4-0-to-4-3/]

Properties

bool	active	`true`
int	audio_max_polyphony	`32`
AnimationCallbackModeDiscrete	callback_mode_discrete	`1`

AnimationCallbackModeMethod	callback_mode_method	`0`
AnimationCallbackModeProcess	callback_mode_process	`1`
bool	deterministic	`false`
bool	reset_on_save	`true`
NodePath	root_motion_track	`NodePath("")`
NodePath	root_node	`NodePath("..")`

Methods

Variant	_post_process_key_value(animation: Animation, track: int, value: Variant, object_id: int, object_sub_idx: int) virtual const
Error	add_animation_library(name: StringName, library: AnimationLibrary)
void	advance(delta: float)
void	capture(name: StringName, duration: float, trans_type: TransitionType = 0, ease_type: EaseType = 0)

void	clear_caches()
StringName	find_animation(animation: Animation) const
StringName	find_animation_library(animation: Animation) const
Animation	get_animation(name: StringName) const
AnimationLibrary	get_animation_library(name: StringName) const
Array[StringName]	get_animation_library_list() const
PackedStringArray	get_animation_list() const
Vector3	get_root_motion_position() const
Vector3	get_root_motion_position_accumulator() const
Quaternion	get_root_motion_rotation() const
Quaternion	get_root_motion_rotation_accumulator() const
Vector3	get_root_motion_scale() const
Vector3	get_root_motion_scale_accumulator() const
bool	has_animation(name: StringName) const

bool	has_animation_library(name: StringName) const
void	remove_animation_library(name: StringName)
void	rename_animation_library(name: StringName, newname: StringName)

Signals

animation_finished(anim_name: StringName) 🔗

Notifies when an animation finished playing.

Note: This signal is not emitted if an animation is looping.

---

animation_libraries_updated() 🔗

Notifies when the animation libraries have changed.

---

animation_list_changed() 🔗

Notifies when an animation list is changed.

---

animation_started(anim_name: StringName) 🔗

Notifies when an animation starts playing.

---

caches_cleared() 🔗

Notifies when the caches have been cleared, either automatically, or manually via clear_caches.

---

mixer_applied() 🔗

Notifies when the blending result related have been applied to the target objects.

---

mixer_updated() 🔗

Notifies when the property related process have been updated.

Enumerations

enum **AnimationCallbackModeProcess**: 🔗

AnimationCallbackModeProcess
ANIMATION_CALLBACK_MODE_PROCESS_PHYSICS = `0`

Process animation during physics frames (see Node.NOTIFICATION_INTERNAL_PHYSICS_PROCESS). This is especially useful when animating physics bodies.

AnimationCallbackModeProcess
ANIMATION_CALLBACK_MODE_PROCESS_IDLE = `1`

Process animation during process frames (see Node.NOTIFICATION_INTERNAL_PROCESS).

AnimationCallbackModeProcess
ANIMATION_CALLBACK_MODE_PROCESS_MANUAL = `2`

Do not process animation. Use advance to process the animation manually.

---

enum **AnimationCallbackModeMethod**: 🔗

AnimationCallbackModeMethod
ANIMATION_CALLBACK_MODE_METHOD_DEFERRED = `0`

Batch method calls during the animation process, then do the calls after events are processed. This avoids bugs involving deleting nodes or modifying the AnimationPlayer while playing.

AnimationCallbackModeMethod
ANIMATION_CALLBACK_MODE_METHOD_IMMEDIATE = `1`

Make method calls immediately when reached in the animation.

---

enum **AnimationCallbackModeDiscrete**: 🔗

AnimationCallbackModeDiscrete
ANIMATION_CALLBACK_MODE_DISCRETE_DOMINANT = `0`

An Animation.UPDATE_DISCRETE track value takes precedence when blending Animation.UPDATE_CONTINUOUS or Animation.UPDATE_CAPTURE track values and Animation.UPDATE_DISCRETE track values.

AnimationCallbackModeDiscrete
ANIMATION_CALLBACK_MODE_DISCRETE_RECESSIVE = `1`

An Animation.UPDATE_CONTINUOUS or Animation.UPDATE_CAPTURE track value takes precedence when blending the Animation.UPDATE_CONTINUOUS or Animation.UPDATE_CAPTURE track values and the Animation.UPDATE_DISCRETE track values. This is the default behavior for AnimationPlayer.

AnimationCallbackModeDiscrete
ANIMATION_CALLBACK_MODE_DISCRETE_FORCE_CONTINUOUS = `2`

Always treat the Animation.UPDATE_DISCRETE track value as Animation.UPDATE_CONTINUOUS with

Animation.INTERPOLATION_NEAREST. This is the default behavior for AnimationTree.

If a value track has non-numeric type key values, it is internally converted to use ANIMATION_CALLBACK_MODE_DISCRETE_RECESSIVE with Animation.UPDATE_DISCRETE.

Property Descriptions

bool **active** = `true` 🔗

- void **set_active**(value: bool)
- bool **is_active**()

If `true`, the **AnimationMixer** will be processing.

---

int **audio_max_polyphony** = `32` 🔗

- void **set_audio_max_polyphony**(value: int)
- int **get_audio_max_polyphony**()

The number of possible simultaneous sounds for each of the assigned AudioStreamPlayers.

For example, if this value is `32` and the animation has two audio tracks, the two AudioStreamPlayers assigned can play simultaneously up to `32` voices each.

AnimationCallbackModeDiscrete **callback_mode_discrete** = `1` 🔗

- void **set_callback_mode_discrete**(value: AnimationCallbackModeDiscrete)
- AnimationCallbackModeDiscrete **get_callback_mode_discrete**()

Ordinarily, tracks can be set to Animation.UPDATE_DISCRETE to update infrequently, usually when using nearest interpolation.

However, when blending with Animation.UPDATE_CONTINUOUS several results are considered. The callback_mode_discrete specify it explicitly. See also AnimationCallbackModeDiscrete.

To make the blended results look good, it is recommended to set this to ANIMATION_CALLBACK_MODE_DISCRETE_FORCE_CONTINUOUS to update every frame during blending. Other values exist for compatibility and they are fine if there is no blending, but not so, may produce artifacts.

AnimationCallbackModeMethod **callback_mode_method** = `0`

- void **set_callback_mode_method**(value: AnimationCallbackModeMethod)
- AnimationCallbackModeMethod **get_callback_mode_method**()

The call mode used for "Call Method" tracks.

AnimationCallbackModeProcess **callback_mode_process** = `1`

- void **set_callback_mode_process**(value: AnimationCallbackModeProcess)
- AnimationCallbackModeProcess **get_callback_mode_process**()

The process notification in which to update animations.

---

bool **deterministic** = `false`

- void **set_deterministic**(value: bool)
- bool **is_deterministic**()

If `true`, the blending uses the deterministic algorithm. The total weight is not normalized and the result is accumulated with an initial value (`0` or a `"RESET"` animation if present).

This means that if the total amount of blending is `0.0`, the result is equal to the `"RESET"` animation.

If the number of tracks between the blended animations is different, the animation with the missing track is treated as if it had the initial value.

If `false`, The blend does not use the deterministic algorithm. The total weight is normalized and always `1.0`. If the number of tracks between the blended animations is different, nothing is done about the animation that is missing a track.

Note: In AnimationTree, the blending with AnimationNodeAdd2, AnimationNodeAdd3, AnimationNodeSub2 or the weight greater than `1.0` may produce unexpected results.

For example, if AnimationNodeAdd2 blends two nodes with the amount `1.0`, then total weight is `2.0` but it will be normalized to make the total amount `1.0` and the result will be equal to AnimationNodeBlend2 with the amount `0.5`.

---

bool **reset_on_save** = `true` 🔗

- void **set_reset_on_save_enabled**(value: bool)
- bool **is_reset_on_save_enabled**()

This is used by the editor. If set to `true`, the scene will be saved with the effects of the reset animation (the animation with the key `"RESET"`) applied as if it had been seeked to time 0, with the editor keeping the values that the scene had before saving.

This makes it more convenient to preview and edit animations in the editor, as changes to the scene will not be saved as long as they are set in the reset animation.

NodePath **root_motion_track** = `NodePath("")` 🔗

- void **set_root_motion_track**(value: NodePath)

- NodePath **get_root_motion_track**()

The path to the Animation track used for root motion. Paths must be valid scene-tree paths to a node, and must be specified starting from the parent node of the node that will reproduce the animation. The root_motion_track uses the same format as Animation.track_set_path, but note that a bone must be specified.

If the track has type Animation.TYPE_POSITION_3D, Animation.TYPE_ROTATION_3D, or Animation.TYPE_SCALE_3D the transformation will be canceled visually, and the animation will appear to stay in place. See also get_root_motion_position, get_root_motion_rotation, get_root_motion_scale, and RootMotionView.

NodePath **root_node** = `NodePath("..")`

- void **set_root_node**(value: NodePath)
- NodePath **get_root_node**()

The node which node path references will travel from.

Method Descriptions

Variant **_post_process_key_value**(animation: Animation, track: int, value: Variant, object_id: int, object_sub_idx: int) virtual const

A virtual function for processing after getting a key during playback.

Error **add_animation_library**(name: StringName, library: AnimationLibrary)

Adds `library` to the animation player, under the key `name`.

AnimationMixer has a global library by default with an empty string as key. For adding an animation to the global library:

GDScript

```
var global_library = mixer.get_animation_library("")
global_library.add_animation("animation_name", animation_resource)
```

void **advance**(delta: float)

Manually advance the animations by the specified time (in seconds).

void **capture**(name: StringName, duration: float, trans_type: TransitionType = 0, ease_type: EaseType = 0)

If the animation track specified by `name` has an option Animation.UPDATE_CAPTURE, stores current values of the objects indicated by the track path as a cache. If there is already a captured cache, the old cache is discarded.

After this it will interpolate with current animation blending result during the playback process for the time specified by `duration`, working like a crossfade.

You can specify `trans_type` as the curve for the interpolation. For better results, it may be appropriate to specify Tween.TRANS_LINEAR for cases where the first key of the track begins with a non-zero value or where the key value does not change, and Tween.TRANS_QUAD for cases where the key value changes linearly.

void **clear_caches**()

AnimationMixer caches animated nodes. It may not notice if a node disappears; clear_caches forces it to update the cache again.

StringName **find_animation**(animation: Animation) const

Returns the key of `animation` or an empty StringName if not found.

StringName **find_animation_library**(animation: Animation) const

Returns the key for the AnimationLibrary that contains `animation` or an empty StringName if not found.

Animation **get_animation**(name: StringName) const

Returns the Animation with the key `name`. If the animation does not exist, `null` is returned and an error is logged.

AnimationLibrary **get_animation_library**(name: StringName) const

Returns the first AnimationLibrary with key `name` or `null` if not found.

To get the **AnimationMixer**'s global animation library, use `get_animation_library("")`.

Array[StringName] **get_animation_library_list**() const

Returns the list of stored library keys.

PackedStringArray **get_animation_list**() const

Returns the list of stored animation keys.

Vector3 **get_root_motion_position**() const

Retrieve the motion delta of position with the root_motion_track as a Vector3 that can be used elsewhere.

If root_motion_track is not a path to a track of type Animation.TYPE_POSITION_3D, returns `Vector3(0, 0, 0)`.

See also root_motion_track and RootMotionView.

The most basic example is applying position to CharacterBody3D:

GDScript

```
var current_rotation: Quaternion

func _process(delta):
    if Input.is_action_just_pressed("animate"):
        current_rotation = get_quaternion()
        state_machine.travel("Animate")
    var velocity: Vector3 = current_rotation *
animation_tree.get_root_motion_position() / delta
    set_velocity(velocity)
    move_and_slide()
```

By using this in combination with get_root_motion_rotation_accumulator, you can apply the root motion position more correctly to account for the rotation of the node.

GDScript

```
func _process(delta):
    if Input.is_action_just_pressed("animate"):
        state_machine.travel("Animate")
    set_quaternion(get_quaternion() *
animation_tree.get_root_motion_rotation())
    var velocity: Vector3 =
(animation_tree.get_root_motion_rotation_accumulator().inverse() *
get_quaternion()) * animation_tree.get_root_motion_position() /
delta
    set_velocity(velocity)
    move_and_slide()
```

Vector3 **get_root_motion_position_accumulator**() const

Retrieve the blended value of the position tracks with the root_motion_track as a Vector3 that can be used elsewhere.

This is useful in cases where you want to respect the initial key values of the animation.

For example, if an animation with only one key `Vector3(0, 0, 0)` is played in the previous frame and then an animation with only one key `Vector3(1, 0, 1)` is played in the next frame, the difference can be calculated as follows:

GDScript

```
var prev_root_motion_position_accumulator: Vector3

func _process(delta):
    if Input.is_action_just_pressed("animate"):
        state_machine.travel("Animate")
    var current_root_motion_position_accumulator: Vector3 =
animation_tree.get_root_motion_position_accumulator()
    var difference: Vector3 =
current_root_motion_position_accumulator -
prev_root_motion_position_accumulator
    prev_root_motion_position_accumulator =
current_root_motion_position_accumulator
    transform.origin += difference
```

However, if the animation loops, an unintended discrete change may occur, so this is only useful for some simple use cases.

Quaternion **get_root_motion_rotation**() const 🔗

Retrieve the motion delta of rotation with the root_motion_track as a Quaternion that can be used elsewhere.

If root_motion_track is not a path to a track of type Animation.TYPE_ROTATION_3D, returns `Quaternion(0, 0, 0, 1)`.

See also root_motion_track and RootMotionView.

The most basic example is applying rotation to CharacterBody3D:

GDScript

```
func _process(delta):
    if Input.is_action_just_pressed("animate"):
        state_machine.travel("Animate")
    set_quaternion(get_quaternion() *
animation_tree.get_root_motion_rotation())
```

Quaternion **get_root_motion_rotation_accumulator**() const

Retrieve the blended value of the rotation tracks with the root_motion_track as a Quaternion that can be used elsewhere.

This is necessary to apply the root motion position correctly, taking rotation into account. See also get_root_motion_position.

Also, this is useful in cases where you want to respect the initial key values of the animation.

For example, if an animation with only one key `Quaternion(0, 0, 0, 1)` is played in the previous frame and then an animation with only one key `Quaternion(0, 0.707, 0, 0.707)` is played in the next frame, the difference can be calculated as follows:

GDScript

```
var prev_root_motion_rotation_accumulator: Quaternion

func _process(delta):
    if Input.is_action_just_pressed("animate"):
        state_machine.travel("Animate")
    var current_root_motion_rotation_accumulator: Quaternion =
animation_tree.get_root_motion_rotation_accumulator()
    var difference: Quaternion =
prev_root_motion_rotation_accumulator.inverse() *
current_root_motion_rotation_accumulator
    prev_root_motion_rotation_accumulator =
current_root_motion_rotation_accumulator
    transform.basis *=  Basis(difference)
```

However, if the animation loops, an unintended discrete change may occur, so this is only useful for some simple use cases.

Vector3 **get_root_motion_scale**() const

Retrieve the motion delta of scale with the root_motion_track as a Vector3 that can be used elsewhere.

If root_motion_track is not a path to a track of type Animation.TYPE_SCALE_3D, returns `Vector3(0, 0, 0)`.

See also root_motion_track and RootMotionView.

The most basic example is applying scale to CharacterBody3D:

GDScript

```
var current_scale: Vector3 = Vector3(1, 1, 1)
var scale_accum: Vector3 = Vector3(1, 1, 1)

func _process(delta):
    if Input.is_action_just_pressed("animate"):
        current_scale = get_scale()
        scale_accum = Vector3(1, 1, 1)
        state_machine.travel("Animate")
    scale_accum += animation_tree.get_root_motion_scale()
    set_scale(current_scale * scale_accum)
```

Vector3 **get_root_motion_scale_accumulator**() const

Retrieve the blended value of the scale tracks with the root_motion_track as a Vector3 that can be used elsewhere.

For example, if an animation with only one key `Vector3(1, 1, 1)` is played in the previous frame and then an animation with only one key `Vector3(2, 2, 2)` is played in the next frame, the difference can be calculated as follows:

GDScript

```
var prev_root_motion_scale_accumulator: Vector3

func _process(delta):
    if Input.is_action_just_pressed("animate"):
        state_machine.travel("Animate")
    var current_root_motion_scale_accumulator: Vector3 =
animation_tree.get_root_motion_scale_accumulator()
    var difference: Vector3 = current_root_motion_scale_accumulator
- prev_root_motion_scale_accumulator
    prev_root_motion_scale_accumulator =
current_root_motion_scale_accumulator
    transform.basis = transform.basis.scaled(difference)
```

However, if the animation loops, an unintended discrete change may occur, so this is only useful for some simple use cases.

bool **has_animation**(name: StringName) const

Returns `true` if the **AnimationMixer** stores an Animation with key `name`.

bool **has_animation_library**(name: StringName) const

Returns `true` if the **AnimationMixer** stores an AnimationLibrary with key `name`.

void **remove_animation_library**(name: StringName)

Removes the AnimationLibrary associated with the key `name`.

void **rename_animation_library**(name: StringName, newname: StringName)

Moves the AnimationLibrary associated with the key `name` to the key `newname`.

AnimationPlayer

Inherits: AnimationMixer **<** Node **<** Object

A node used for animation playback.

Description

An animation player is used for general-purpose playback of animations. It contains a dictionary of AnimationLibrary resources and custom blend times between animation transitions.

Some methods and properties use a single key to reference an animation directly. These keys are formatted as the key for the library, followed by a forward slash, then the key for the animation within the library, for example `"movement/run"`. If the library's key is an empty string (known as the default library), the forward slash is omitted, being the same key used by the library.

AnimationPlayer is better-suited than Tween for more complex animations, for example ones with non-trivial timings. It can also be used over Tween if the animation track editor is more convenient than doing it in code.

Updating the target properties of animations occurs at the process frame.

Tutorials

- 2D Sprite animation
- Animation documentation index

- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

String	assigned_animation	
String	autoplay	`""`
String	current_animation	`""`
float	current_animation_length	
float	current_animation_position	
bool	movie_quit_on_finish	`false`
bool	playback_auto_capture	`true`
float	playback_auto_capture_duration	`-1.0`
EaseType	playback_auto_capture_ease_type	`0`
TransitionType	playback_auto_capture_transition_type	`0`

float	playback_default_blend_time	`0.0`
float	speed_scale	`1.0`

Methods

StringName	animation_get_next(animation_from: StringName) const
void	animation_set_next(animation_from: StringName, animation_to: StringName)
void	clear_queue()
float	get_blend_time(animation_from: StringName, animation_to: StringName) const
AnimationMethodCallMode	get_method_call_mode() const
float	get_playing_speed() const
AnimationProcessCallback	get_process_callback() const

PackedStringArray	get_queue()
NodePath	get_root() const
bool	is_playing() const
void	pause()
void	play(name: StringName = &"", custom_blend: float = -1, custom_speed: float = 1.0, from_end: bool = false)
void	play_backwards(name: StringName = &"", custom_blend: float = -1)
void	play_with_capture(name: StringName = &"", duration: float = -1.0, custom_blend: float = -1, custom_speed: float = 1.0, from_end: bool = false, trans_type: TransitionType = 0, ease_type: EaseType = 0)
void	queue(name: StringName)
void	seek(seconds: float, update: bool = false, update_only: bool = false)

void	set_blend_time(animation_from: StringName, animation_to: StringName, sec: float)
void	set_method_call_mode(mode: AnimationMethodCallMode)
void	set_process_callback(mode: AnimationProcessCallback)
void	set_root(path: NodePath)
void	stop(keep_state: bool = false)

Signals

animation_changed(old_name: StringName, new_name: StringName) 🔗

Emitted when a queued animation plays after the previous animation finished. See also queue.

Note: The signal is not emitted when the animation is changed via play or by an AnimationTree.

current_animation_changed(name: String) 🔗

Emitted when current_animation changes.

Enumerations

enum **AnimationProcessCallback**: 🔗

AnimationProcessCallback **ANIMATION_PROCESS_PHYSICS** = `0`

Deprecated: See AnimationMixer.ANIMATION_CALLBACK_MODE_PROCESS_PHYSICS.

AnimationProcessCallback **ANIMATION_PROCESS_IDLE** = `1`

Deprecated: See AnimationMixer.ANIMATION_CALLBACK_MODE_PROCESS_IDLE.

AnimationProcessCallback **ANIMATION_PROCESS_MANUAL** = `2`

Deprecated: See AnimationMixer.ANIMATION_CALLBACK_MODE_PROCESS_MANUAL.

---

enum **AnimationMethodCallMode**: 🔗

AnimationMethodCallMode **ANIMATION_METHOD_CALL_DEFERRED** = `0`

Deprecated: See AnimationMixer.ANIMATION_CALLBACK_MODE_METHOD_DEFERRED.

AnimationMethodCallMode **ANIMATION_METHOD_CALL_IMMEDIATE** = `1`

Deprecated: See AnimationMixer.ANIMATION_CALLBACK_MODE_METHOD_IMMEDIATE.

Property Descriptions

String **assigned_animation**

- void **set_assigned_animation**(value: String)
- String **get_assigned_animation**()

If playing, the current animation's key, otherwise, the animation last played. When set, this changes the animation, but will not play it unless already playing. See also current_animation.

String **autoplay** = `""`

- void **set_autoplay**(value: String)
- String **get_autoplay**()

The key of the animation to play when the scene loads.

String **current_animation** = `""`

- void **set_current_animation**(value: String)
- String **get_current_animation**()

The key of the currently playing animation. If no animation is playing, the property's value is an empty string. Changing this value does not restart the animation. See play for more information on playing animations.

Note: While this property appears in the Inspector, it's not meant to be edited, and it's not saved in the scene. This property is mainly used to get the currently playing animation, and internally for animation playback tracks. For more information, see Animation.

---

float **current_animation_length**

- float **get_current_animation_length**()

The length (in seconds) of the currently playing animation.

float **current_animation_position**

- float **get_current_animation_position**()

The position (in seconds) of the currently playing animation.

bool **movie_quit_on_finish** = `false`

- void **set_movie_quit_on_finish_enabled**(value: bool)
- bool **is_movie_quit_on_finish_enabled**()

If `true` and the engine is running in Movie Maker mode (see MovieWriter), exits the engine with SceneTree.quit as soon as an animation is done playing in this **AnimationPlayer**. A message is printed when the engine quits for this reason.

Note: This obeys the same logic as the AnimationMixer.animation_finished signal, so it will not quit the engine if the animation is set to be looping.

bool **playback_auto_capture** = `true`

- void **set_auto_capture**(value: bool)
- bool **is_auto_capture**()

If `true`, performs AnimationMixer.capture before playback automatically. This means just play_with_capture is executed with default arguments instead of play.

Note: Capture interpolation is only performed if the animation contains a capture track. See also Animation.UPDATE_CAPTURE.

float **playback_auto_capture_duration** = `-1.0`

- void **set_auto_capture_duration**(value: float)
- float **get_auto_capture_duration**()

See also play_with_capture and AnimationMixer.capture.

If playback_auto_capture_duration is negative value, the duration is set to the interval between the current position and the first key.

EaseType **playback_auto_capture_ease_type** = `0`

- void **set_auto_capture_ease_type**(value: EaseType)
- EaseType **get_auto_capture_ease_type**()

The ease type of the capture interpolation. See also EaseType.

TransitionType **playback_auto_capture_transition_type** = `0`

- void **set_auto_capture_transition_type**(value: TransitionType)
- TransitionType **get_auto_capture_transition_type**()

The transition type of the capture interpolation. See also TransitionType.

float **playback_default_blend_time** = `0.0`

- void **set_default_blend_time**(value: float)
- float **get_default_blend_time**()

The default time in which to blend animations. Ranges from 0 to 4096 with 0.01 precision.

---

float **speed_scale** = `1.0` 🔗

- void **set_speed_scale**(value: float)
- float **get_speed_scale**()

The speed scaling ratio. For example, if this value is `1`, then the animation plays at normal speed. If it's `0.5`, then it plays at half speed. If it's `2`, then it plays at double speed.

If set to a negative value, the animation is played in reverse. If set to `0`, the animation will not advance.

Method Descriptions

StringName **animation_get_next**(animation_from: StringName) const 🔗

Returns the key of the animation which is queued to play after the `animation_from` animation.

---

void **animation_set_next**(animation_from: StringName, animation_to: StringName) 🔗

Triggers the `animation_to` animation when the `animation_from` animation completes.

---

void **clear_queue**() 🔗

Clears all queued, unplayed animations.

float **get_blend_time**(animation_from: StringName, animation_to: StringName) const 🔗

Returns the blend time (in seconds) between two animations, referenced by their keys.

AnimationMethodCallMode **get_method_call_mode**() const 🔗

Deprecated: Use AnimationMixer.callback_mode_method instead.

Returns the call mode used for "Call Method" tracks.

float **get_playing_speed**() const 🔗

Returns the actual playing speed of current animation or `0` if not playing. This speed is the speed_scale property multiplied by `custom_speed` argument specified when calling the play method.

Returns a negative value if the current animation is playing backwards.

AnimationProcessCallback **get_process_callback**() const 🔗

Deprecated: Use AnimationMixer.callback_mode_process instead.

Returns the process notification in which to update animations.

PackedStringArray **get_queue**() 🔗

Returns a list of the animation keys that are currently queued to play.

NodePath **get_root**() const 🔗

Deprecated: Use AnimationMixer.root_node instead.

Returns the node which node path references will travel from.

---

bool **is_playing**() const 🔗

Returns `true` if an animation is currently playing (even if speed_scale and/or `custom_speed` are `0`).

---

void **pause**() 🔗

Pauses the currently playing animation. The current_animation_position will be kept and calling play or play_backwards without arguments or with the same animation name as assigned_animation will resume the animation.

See also stop.

---

void **play**(name: StringName = &"", custom_blend: float = -1, custom_speed: float = 1.0, from_end: bool = false) 🔗

Plays the animation with key `name`. Custom blend times and speed can be set.

The `from_end` option only affects when switching to a new animation track, or if the same track but at the start or end. It does not affect resuming playback that was paused in the middle of an animation. If `custom_speed` is negative and `from_end` is `true`, the animation will play backwards (which is equivalent to calling play_backwards).

The **AnimationPlayer** keeps track of its current or last played animation with assigned_animation. If this method is called with that same animation `name`, or with no `name` parameter, the assigned animation will resume playing if it was paused.

Note: The animation will be updated the next time the **AnimationPlayer** is processed. If other variables are updated at the same time this is called, they may be updated too early. To perform the update immediately, call `advance(0)`.

---

void **play_backwards**(name: StringName = &"", custom_blend: float = -1) 🔗

Plays the animation with key `name` in reverse.

This method is a shorthand for play with `custom_speed = -1.0` and `from_end = true`, so see its description for more information.

---

void **play_with_capture**(name: StringName = &"", duration: float = -1.0, custom_blend: float = -1, custom_speed: float = 1.0, from_end: bool = false, trans_type: TransitionType = 0, ease_type: EaseType = 0) 🔗

See also AnimationMixer.capture.

You can use this method to use more detailed options for capture than those performed by playback_auto_capture. When playback_auto_capture is `false`, this method is almost the same as the following:

```
capture(name, duration, trans_type, ease_type)
```

```
play(name, custom_blend, custom_speed, from_end)
```

If `name` is blank, it specifies assigned_animation.

If `duration` is a negative value, the duration is set to the interval between the current position and the first key, when `from_end` is `true`, uses the interval between the current position and the last key instead.

Note: The `duration` takes speed_scale into account, but `custom_speed` does not, because the capture cache is interpolated with the blend result and the result may contain multiple animations.

void **queue**(name: StringName)

Queues an animation for playback once the current animation and all previously queued animations are done.

Note: If a looped animation is currently playing, the queued animation will never play unless the looped animation is stopped somehow.

void **seek**(seconds: float, update: bool = false, update_only: bool = false)

Seeks the animation to the `seconds` point in time (in seconds). If `update` is `true`, the animation updates too, otherwise it updates at process time. Events between the current frame and `seconds` are skipped.

If `update_only` is `true`, the method / audio / animation playback tracks will not be processed.

Note: Seeking to the end of the animation doesn't emit AnimationMixer.animation_finished. If you want to skip animation and emit the signal, use AnimationMixer.advance.

---

void **set_blend_time**(animation_from: StringName, animation_to: StringName, sec: float)

Specifies a blend time (in seconds) between two animations, referenced by their keys.

---

void **set_method_call_mode**(mode: AnimationMethodCallMode)

Deprecated: Use AnimationMixer.callback_mode_method instead.

Sets the call mode used for "Call Method" tracks.

---

void **set_process_callback**(mode: AnimationProcessCallback)

Deprecated: Use AnimationMixer.callback_mode_process instead.

Sets the process notification in which to update animations.

---

void **set_root**(path: NodePath)

Deprecated: Use AnimationMixer.root_node instead.

Sets the node which node path references will travel from.

---

void **stop**(keep_state: bool = false)

AnimationTree

Inherits: AnimationMixer **<** Node **<** Object

A node used for advanced animation transitions in an AnimationPlayer.

Description

A node used for advanced animation transitions in an AnimationPlayer.

Note: When linked with an AnimationPlayer, several properties and methods of the corresponding AnimationPlayer will not function as expected. Playback and transitions should be handled using only the **AnimationTree** and its constituent AnimationNode(s). The AnimationPlayer node should be used solely for adding, deleting, and editing animations.

Tutorials

- Using AnimationTree
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

NodePath	advance_expression_base_node	`NodePath(".")`
NodePath	anim_player	`NodePath("")`
AnimationCallbackModeDiscrete	callback_mode_discrete	`2` (overrides AnimationMixer)
bool	deterministic	`true` (overrides AnimationMixer)
AnimationRootNode	tree_root	

Methods

AnimationProcessCallback	get_process_callback() const
void	set_process_callback(mode: AnimationProcessCallback)

Signals

animation_player_changed() 🔗

Emitted when the anim_player is changed.

Enumerations

enum **AnimationProcessCallback**: 🔗

AnimationProcessCallback **ANIMATION_PROCESS_PHYSICS** = `0`

Deprecated: See AnimationMixer.ANIMATION_CALLBACK_MODE_PROCESS_PHYSICS.

AnimationProcessCallback **ANIMATION_PROCESS_IDLE** = `1`

Deprecated: See AnimationMixer.ANIMATION_CALLBACK_MODE_PROCESS_IDLE.

AnimationProcessCallback **ANIMATION_PROCESS_MANUAL** = `2`

Deprecated: See AnimationMixer.ANIMATION_CALLBACK_MODE_PROCESS_MANUAL.

Property Descriptions

NodePath **advance_expression_base_node** = `NodePath(".")`

- void **set_advance_expression_base_node**(value: NodePath)
- NodePath **get_advance_expression_base_node**()

The path to the Node used to evaluate the AnimationNode Expression if one is not explicitly specified internally.

NodePath **anim_player** = `NodePath("")`

- void **set_animation_player**(value: NodePath)
- NodePath **get_animation_player**()

The path to the AnimationPlayer used for animating.

AnimationRootNode **tree_root**

- void **set_tree_root**(value: AnimationRootNode)
- AnimationRootNode **get_tree_root**()

The root animation node of this **AnimationTree**. See AnimationRootNode.

Method Descriptions

AnimationProcessCallback **get_process_callback**() const

Deprecated: Use AnimationMixer.callback_mode_process instead.

Returns the process notification in which to update animations.

---

void **set_process_callback**(mode: AnimationProcessCallback)

Deprecated: Use AnimationMixer.callback_mode_process instead.

Sets the process notification in which to update animations.

Area2D

Inherits: CollisionObject2D **<** Node2D **<** CanvasItem **<** Node **<** Object

A region of 2D space that detects other CollisionObject2Ds entering or exiting it.

Description

Area2D is a region of 2D space defined by one or multiple CollisionShape2D or CollisionPolygon2D child nodes. It detects when other CollisionObject2Ds enter or exit it, and it also keeps track of which collision objects haven't exited it yet (i.e. which one are overlapping it).

This node can also locally alter or override physics parameters (gravity, damping) and route audio to custom audio buses.

Note: Areas and bodies created with PhysicsServer2D might not interact as expected with **Area2D**s, and might not emit signals or track objects correctly.

Tutorials

- Using Area2D
- 2D Dodge The Creeps Demo [https://godotengine.org/asset-library/asset/2712]
- 2D Pong Demo [https://godotengine.org/asset-library/asset/2728]
- 2D Platformer Demo [https://godotengine.org/asset-library/asset/2727]

Properties

float	angular_damp	`1.0`
SpaceOverride	angular_damp_space_override	`0`
StringName	audio_bus_name	`&"Master"`
bool	audio_bus_override	`false`
float	gravity	`980.0`
Vector2	gravity_direction	`Vector2(0, 1)`
bool	gravity_point	`false`
Vector2	gravity_point_center	`Vector2(0, 1)`
float	gravity_point_unit_distance	`0.0`
SpaceOverride	gravity_space_override	`0`
float	linear_damp	`0.1`
SpaceOverride	linear_damp_space_override	`0`

bool	monitorable	`true`
bool	monitoring	`true`
int	priority	`0`

Methods

Array[Area2D]	get_overlapping_areas() const
Array[Node2D]	get_overlapping_bodies() const
bool	has_overlapping_areas() const
bool	has_overlapping_bodies() const
bool	overlaps_area(area: Node) const
bool	overlaps_body(body: Node) const

Signals

area_entered(area: Area2D) 🔗

Emitted when the received `area` enters this area. Requires monitoring to be set to `true`.

area_exited(area: Area2D) 🔗

Emitted when the received `area` exits this area. Requires monitoring to be set to `true`.

area_shape_entered(area_rid: RID, area: Area2D, area_shape_index: int, local_shape_index: int) 🔗

Emitted when a Shape2D of the received `area` enters a shape of this area. Requires monitoring to be set to `true`.

`local_shape_index` and `area_shape_index` contain indices of the interacting shapes from this area and the other area, respectively. `area_rid` contains the RID of the other area. These values can be used with the PhysicsServer2D.

Example of getting the CollisionShape2D node from the shape index:

GDScript

```
var other_shape_owner = area.shape_find_owner(area_shape_index)
var other_shape_node =
area.shape_owner_get_owner(other_shape_owner)

var local_shape_owner = shape_find_owner(local_shape_index)
var local_shape_node = shape_owner_get_owner(local_shape_owner)
```

area_shape_exited(area_rid: RID, area: Area2D, area_shape_index: int, local_shape_index: int) 🔗

Emitted when a Shape2D of the received `area` exits a shape of this area. Requires monitoring to be set to `true`.

See also area_shape_entered.

---

body_entered(body: Node2D)

Emitted when the received `body` enters this area. `body` can be a PhysicsBody2D or a TileMap. TileMaps are detected if their TileSet has collision shapes configured. Requires monitoring to be set to `true`.

---

body_exited(body: Node2D)

Emitted when the received `body` exits this area. `body` can be a PhysicsBody2D or a TileMap. TileMaps are detected if their TileSet has collision shapes configured. Requires monitoring to be set to `true`.

---

body_shape_entered(body_rid: RID, body: Node2D, body_shape_index: int, local_shape_index: int)

Emitted when a Shape2D of the received `body` enters a shape of this area. `body` can be a PhysicsBody2D or a TileMap. TileMaps are detected if their TileSet has collision shapes configured. Requires monitoring to be set to `true`.

`local_shape_index` and `body_shape_index` contain indices of the interacting shapes from this area and the interacting body, respectively. `body_rid` contains the RID of the body. These values can be used with the PhysicsServer2D.

Example of getting the CollisionShape2D node from the shape index:

GDScript

```gdscript
var body_shape_owner = body.shape_find_owner(body_shape_index)
var body_shape_node = body.shape_owner_get_owner(body_shape_owner)

var local_shape_owner = shape_find_owner(local_shape_index)
var local_shape_node = shape_owner_get_owner(local_shape_owner)
```

body_shape_exited(body_rid: RID, body: Node2D, body_shape_index: int, local_shape_index: int)

Emitted when a Shape2D of the received `body` exits a shape of this area. `body` can be a PhysicsBody2D or a TileMap. TileMaps are detected if their TileSet has collision shapes configured. Requires monitoring to be set to `true`.

See also body_shape_entered.

Enumerations

enum **SpaceOverride**:

SpaceOverride **SPACE_OVERRIDE_DISABLED** = `0`

This area does not affect gravity/damping.

SpaceOverride **SPACE_OVERRIDE_COMBINE** = `1`

This area adds its gravity/damping values to whatever has been calculated so far (in priority order).

SpaceOverride **SPACE_OVERRIDE_COMBINE_REPLACE** = `2`

This area adds its gravity/damping values to whatever has been calculated so far (in priority order), ignoring any lower priority areas.

SpaceOverride **SPACE_OVERRIDE_REPLACE** = `3`

This area replaces any gravity/damping, even the defaults, ignoring any lower priority areas.

SpaceOverride **SPACE_OVERRIDE_REPLACE_COMBINE** = `4`

This area replaces any gravity/damping calculated so far (in priority order), but keeps calculating the rest of the areas.

Property Descriptions

float **angular_damp** = `1.0` 🔗

- void **set_angular_damp**(value: float)
- float **get_angular_damp**()

The rate at which objects stop spinning in this area. Represents the angular velocity lost per second.

See ProjectSettings.physics/2d/default_angular_damp for more details about damping.

SpaceOverride **angular_damp_space_override** = `0` 🔗

- void **set_angular_damp_space_override_mode**(value: SpaceOverride)
- SpaceOverride **get_angular_damp_space_override_mode**()

Override mode for angular damping calculations within this area. See SpaceOverride for possible values.

StringName **audio_bus_name** = `&"Master"`

- void **set_audio_bus_name**(value: StringName)
- StringName **get_audio_bus_name**()

The name of the area's audio bus.

---

bool **audio_bus_override** = `false`

- void **set_audio_bus_override**(value: bool)
- bool **is_overriding_audio_bus**()

If `true`, the area's audio bus overrides the default audio bus.

---

float **gravity** = `980.0`

- void **set_gravity**(value: float)
- float **get_gravity**()

The area's gravity intensity (in pixels per second squared). This value multiplies the gravity direction. This is useful to alter the force of gravity without altering its direction.

Vector2 **gravity_direction** = `Vector2(0, 1)`

- void **set_gravity_direction**(value: Vector2)
- Vector2 **get_gravity_direction**()

The area's gravity vector (not normalized).

---

bool **gravity_point** = `false` 🔗

- void **set_gravity_is_point**(value: bool)
- bool **is_gravity_a_point**()

If `true`, gravity is calculated from a point (set via gravity_point_center). See also gravity_space_override.

Vector2 **gravity_point_center** = `Vector2(0, 1)` 🔗

- void **set_gravity_point_center**(value: Vector2)
- Vector2 **get_gravity_point_center**()

If gravity is a point (see gravity_point), this will be the point of attraction.

---

float **gravity_point_unit_distance** = `0.0` 🔗

- void **set_gravity_point_unit_distance**(value: float)
- float **get_gravity_point_unit_distance**()

The distance at which the gravity strength is equal to gravity. For example, on a planet 100 pixels in radius with a surface gravity of 4.0 px/s², set the gravity to 4.0 and the unit distance to 100.0. The gravity will have falloff according to the inverse square law, so in the example, at 200 pixels from the center the gravity will be 1.0 px/s² (twice the distance, 1/4th the gravity), at 50 pixels it will be 16.0 px/s² (half the distance, 4x the gravity), and so on.

The above is true only when the unit distance is a positive number. When this is set to 0.0, the gravity will be constant regardless of distance.

SpaceOverride **gravity_space_override** = `0` 🔗

- void **set_gravity_space_override_mode**(value: SpaceOverride)
- SpaceOverride **get_gravity_space_override_mode**()

Override mode for gravity calculations within this area. See SpaceOverride for possible values.

float **linear_damp** = `0.1`

- void **set_linear_damp**(value: float)
- float **get_linear_damp**()

The rate at which objects stop moving in this area. Represents the linear velocity lost per second.

See ProjectSettings.physics/2d/default_linear_damp for more details about damping.

SpaceOverride **linear_damp_space_override** = `0`

- void **set_linear_damp_space_override_mode**(value: SpaceOverride)
- SpaceOverride **get_linear_damp_space_override_mode**()

Override mode for linear damping calculations within this area. See SpaceOverride for possible values.

bool **monitorable** = `true`

- void **set_monitorable**(value: bool)
- bool **is_monitorable**()

If `true`, other monitoring areas can detect this area.

bool **monitoring** = `true`

- void **set_monitoring**(value: bool)
- bool **is_monitoring**()

If `true`, the area detects bodies or areas entering and exiting it.

---

int **priority** = `0`

- void **set_priority**(value: int)
- int **get_priority**()

The area's priority. Higher priority areas are processed first. The World2D's physics is always processed last, after all areas.

Method Descriptions

Array[Area2D] **get_overlapping_areas**() const

Returns a list of intersecting **Area2D**s. The overlapping area's CollisionObject2D.collision_layer must be part of this area's CollisionObject2D.collision_mask in order to be detected.

For performance reasons (collisions are all processed at the same time) this list is modified once during the physics step, not immediately after objects are moved. Consider using signals instead.

Array[Node2D] **get_overlapping_bodies**() const

Returns a list of intersecting PhysicsBody2Ds and TileMaps. The overlapping body's CollisionObject2D.collision_layer must be part of this area's CollisionObject2D.collision_mask in order to be detected.

For performance reasons (collisions are all processed at the same time) this list is modified once during the physics step, not immediately after objects are moved. Consider using signals instead.

---

bool **has_overlapping_areas**() const 🔗

Returns `true` if intersecting any **Area2D**s, otherwise returns `false`. The overlapping area's CollisionObject2D.collision_layer must be part of this area's CollisionObject2D.collision_mask in order to be detected.

For performance reasons (collisions are all processed at the same time) the list of overlapping areas is modified once during the physics step, not immediately after objects are moved. Consider using signals instead.

---

bool **has_overlapping_bodies**() const 🔗

Returns `true` if intersecting any PhysicsBody2Ds or TileMaps, otherwise returns `false`. The overlapping body's CollisionObject2D.collision_layer must be part of this area's CollisionObject2D.collision_mask in order to be detected.

For performance reasons (collisions are all processed at the same time) the list of overlapping bodies is modified once during the physics step, not immediately after objects are moved. Consider using signals instead.

---

bool **overlaps_area**(area: Node) const 🔗

Returns `true` if the given **Area2D** intersects or overlaps this **Area2D**, `false` otherwise.

Note: The result of this test is not immediate after moving objects. For performance, the list of overlaps is updated once per frame and before the physics step. Consider using signals instead.

---

bool **overlaps_body**(body: Node) const 🔗

Returns `true` if the given physics body intersects or overlaps this **Area2D**, `false` otherwise.

Note: The result of this test is not immediate after moving objects. For performance, list of overlaps is updated once per frame and before the physics step. Consider using signals instead.

The `body` argument can either be a PhysicsBody2D or a TileMap instance. While TileMaps are not physics bodies themselves, they register their tiles with collision shapes as a virtual physics body.

Area3D

Inherits: CollisionObject3D **<** Node3D **<** Node **<** Object

A region of 3D space that detects other CollisionObject3Ds entering or exiting it.

Description

Area3D is a region of 3D space defined by one or multiple CollisionShape3D or CollisionPolygon3D child nodes. It detects when other CollisionObject3Ds enter or exit it, and it also keeps track of which collision objects haven't exited it yet (i.e. which one are overlapping it).

This node can also locally alter or override physics parameters (gravity, damping) and route audio to custom audio buses.

Note: Areas and bodies created with PhysicsServer3D might not interact as expected with **Area3D**s, and might not emit signals or track objects correctly.

Warning: Using a ConcavePolygonShape3D inside a CollisionShape3D child of this node (created e.g. by using the **Create Trimesh Collision Sibling** option in the **Mesh** menu that appears when selecting a MeshInstance3D node) may give unexpected results, since this collision shape is hollow. If this is not desired, it has to be split into multiple ConvexPolygonShape3Ds or primitive shapes like BoxShape3D, or in some cases it may be replaceable by a CollisionPolygon3D.

Tutorials

- Using Area2D
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]
- GUI in 3D Viewport Demo [https://godotengine.org/asset-library/asset/2807]

Properties

float	angular_damp	`0.1`
SpaceOverride	angular_damp_space_override	`0`
StringName	audio_bus_name	`&"Master"`
bool	audio_bus_override	`false`
float	gravity	`9.8`
Vector3	gravity_direction	`Vector3(0, -1, 0)`
bool	gravity_point	`false`
Vector3	gravity_point_center	`Vector3(0, -1, 0)`
float	gravity_point_unit_distance	`0.0`
SpaceOverride	gravity_space_override	`0`

float	linear_damp	`0.1`
SpaceOverride	linear_damp_space_override	`0`
bool	monitorable	`true`
bool	monitoring	`true`
int	priority	`0`
float	reverb_bus_amount	`0.0`
bool	reverb_bus_enabled	`false`
StringName	reverb_bus_name	`&"Master"`
float	reverb_bus_uniformity	`0.0`
float	wind_attenuation_factor	`0.0`
float	wind_force_magnitude	`0.0`
NodePath	wind_source_path	`NodePath("")`

Methods

Array[Area3D]	get_overlapping_areas() const
Array[Node3D]	get_overlapping_bodies() const
bool	has_overlapping_areas() const
bool	has_overlapping_bodies() const
bool	overlaps_area(area: Node) const
bool	overlaps_body(body: Node) const

Signals

area_entered(area: Area3D) 🔗

Emitted when the received `area` enters this area. Requires monitoring to be set to `true`.

area_exited(area: Area3D) 🔗

Emitted when the received `area` exits this area. Requires monitoring to be set to `true`.

area_shape_entered(area_rid: RID, area: Area3D, area_shape_index: int, local_shape_index: int) 🔗

Emitted when a Shape3D of the received `area` enters a shape of this area. Requires monitoring to be set to `true`.

`local_shape_index` and `area_shape_index` contain indices of the interacting shapes from this area and the other area, respectively. `area_rid` contains the RID of the other area. These values can be used with the PhysicsServer3D.

Example of getting the CollisionShape3D node from the shape index:

GDScript

```
var other_shape_owner = area.shape_find_owner(area_shape_index)
var other_shape_node =
area.shape_owner_get_owner(other_shape_owner)

var local_shape_owner = shape_find_owner(local_shape_index)
var local_shape_node = shape_owner_get_owner(local_shape_owner)
```

---

area_shape_exited(area_rid: RID, area: Area3D, area_shape_index: int, local_shape_index: int) 🔗

Emitted when a Shape3D of the received `area` exits a shape of this area. Requires monitoring to be set to `true`.

See also area_shape_entered.

---

body_entered(body: Node3D) 🔗

Emitted when the received `body` enters this area. `body` can be a PhysicsBody3D or a GridMap. GridMaps are detected if their

MeshLibrary has collision shapes configured. Requires monitoring to be set to `true`.

---

body_exited(body: Node3D)

Emitted when the received `body` exits this area. `body` can be a PhysicsBody3D or a GridMap. GridMaps are detected if their MeshLibrary has collision shapes configured. Requires monitoring to be set to `true`.

---

body_shape_entered(body_rid: RID, body: Node3D, body_shape_index: int, local_shape_index: int)

Emitted when a Shape3D of the received `body` enters a shape of this area. `body` can be a PhysicsBody3D or a GridMap. GridMaps are detected if their MeshLibrary has collision shapes configured. Requires monitoring to be set to `true`.

`local_shape_index` and `body_shape_index` contain indices of the interacting shapes from this area and the interacting body, respectively. `body_rid` contains the RID of the body. These values can be used with the PhysicsServer3D.

Example of getting the CollisionShape3D node from the shape index:

GDScript

```
var body_shape_owner = body.shape_find_owner(body_shape_index)
var body_shape_node =
body.shape_owner_get_owner(body_shape_owner)

var local_shape_owner = shape_find_owner(local_shape_index)
var local_shape_node = shape_owner_get_owner(local_shape_owner)
```

---

body_shape_exited(body_rid: RID, body: Node3D, body_shape_index: int, local_shape_index: int) 🔗

Emitted when a Shape3D of the received `body` exits a shape of this area. `body` can be a PhysicsBody3D or a GridMap. GridMaps are detected if their MeshLibrary has collision shapes configured. Requires monitoring to be set to `true`.

See also body_shape_entered.

Enumerations

enum **SpaceOverride**: 🔗

SpaceOverride **SPACE_OVERRIDE_DISABLED** = `0`

This area does not affect gravity/damping.

SpaceOverride **SPACE_OVERRIDE_COMBINE** = `1`

This area adds its gravity/damping values to whatever has been calculated so far (in priority order).

SpaceOverride **SPACE_OVERRIDE_COMBINE_REPLACE** = `2`

This area adds its gravity/damping values to whatever has been calculated so far (in priority order), ignoring any lower priority areas.

SpaceOverride **SPACE_OVERRIDE_REPLACE** = `3`

This area replaces any gravity/damping, even the defaults, ignoring any lower priority areas.

SpaceOverride **SPACE_OVERRIDE_REPLACE_COMBINE** = `4`

This area replaces any gravity/damping calculated so far (in priority order), but keeps calculating the rest of the areas.

Property Descriptions

float **angular_damp** = `0.1`

- void **set_angular_damp**(value: float)
- float **get_angular_damp**()

The rate at which objects stop spinning in this area. Represents the angular velocity lost per second.

See ProjectSettings.physics/3d/default_angular_damp for more details about damping.

SpaceOverride **angular_damp_space_override** = `0`

- void **set_angular_damp_space_override_mode**(value: SpaceOverride)
- SpaceOverride **get_angular_damp_space_override_mode**()

Override mode for angular damping calculations within this area. See SpaceOverride for possible values.

StringName **audio_bus_name** = `&"Master"`

- void **set_audio_bus_name**(value: StringName)
- StringName **get_audio_bus_name**()

The name of the area's audio bus.

---

bool **audio_bus_override** = `false`

- void **set_audio_bus_override**(value: bool)
- bool **is_overriding_audio_bus**()

If `true`, the area's audio bus overrides the default audio bus.

---

float **gravity** = `9.8`

- void **set_gravity**(value: float)
- float **get_gravity**()

The area's gravity intensity (in meters per second squared). This value multiplies the gravity direction. This is useful to alter the force of gravity without altering its direction.

Vector3 **gravity_direction** = `Vector3(0, -1, 0)`

- void **set_gravity_direction**(value: Vector3)
- Vector3 **get_gravity_direction**()

The area's gravity vector (not normalized).

---

bool **gravity_point** = `false`

- void **set_gravity_is_point**(value: bool)
- bool **is_gravity_a_point**()

If `true`, gravity is calculated from a point (set via gravity_point_center). See also gravity_space_override.

Vector3 **gravity_point_center** = `Vector3(0, -1, 0)`

- void **set_gravity_point_center**(value: Vector3)
- Vector3 **get_gravity_point_center**()

If gravity is a point (see gravity_point), this will be the point of attraction.

---

float **gravity_point_unit_distance** = `0.0` 🔗

- void **set_gravity_point_unit_distance**(value: float)
- float **get_gravity_point_unit_distance**()

The distance at which the gravity strength is equal to gravity. For example, on a planet 100 meters in radius with a surface gravity of 4.0 m/s², set the gravity to 4.0 and the unit distance to 100.0. The gravity will have falloff according to the inverse square law, so in the example, at 200 meters from the center the gravity will be 1.0 m/s² (twice the distance, 1/4th the gravity), at 50 meters it will be 16.0 m/s² (half the distance, 4x the gravity), and so on.

The above is true only when the unit distance is a positive number. When this is set to 0.0, the gravity will be constant regardless of distance.

SpaceOverride **gravity_space_override** = `0` 🔗

- void **set_gravity_space_override_mode**(value: SpaceOverride)
- SpaceOverride **get_gravity_space_override_mode**()

Override mode for gravity calculations within this area. See SpaceOverride for possible values.

---

float **linear_damp** = `0.1` 🔗

- void **set_linear_damp**(value: float)
- float **get_linear_damp**()

The rate at which objects stop moving in this area. Represents the linear velocity lost per second.

See ProjectSettings.physics/3d/default_linear_damp for more details about damping.

SpaceOverride **linear_damp_space_override** = `0` 🔗

- void **set_linear_damp_space_override_mode**(value: SpaceOverride)
- SpaceOverride **get_linear_damp_space_override_mode**()

Override mode for linear damping calculations within this area. See SpaceOverride for possible values.

---

bool **monitorable** = `true` 🔗

- void **set_monitorable**(value: bool)
- bool **is_monitorable**()

If `true`, other monitoring areas can detect this area.

---

bool **monitoring** = `true` 🔗

- void **set_monitoring**(value: bool)
- bool **is_monitoring**()

If `true`, the area detects bodies or areas entering and exiting it.

---

int **priority** = `0` 🔗

- void **set_priority**(value: int)
- int **get_priority**()

The area's priority. Higher priority areas are processed first. The World3D's physics is always processed last, after all areas.

---

float **reverb_bus_amount** = `0.0` 🔗

- void **set_reverb_amount**(value: float)
- float **get_reverb_amount**()

The degree to which this area applies reverb to its associated audio. Ranges from `0` to `1` with `0.1` precision.

---

bool **reverb_bus_enabled** = `false` 🔗

- void **set_use_reverb_bus**(value: bool)
- bool **is_using_reverb_bus**()

If `true`, the area applies reverb to its associated audio.

StringName **reverb_bus_name** = `&"Master"` 🔗

- void **set_reverb_bus_name**(value: StringName)
- StringName **get_reverb_bus_name**()

The name of the reverb bus to use for this area's associated audio.

---

float **reverb_bus_uniformity** = `0.0` 🔗

- void **set_reverb_uniformity**(value: float)
- float **get_reverb_uniformity**()

The degree to which this area's reverb is a uniform effect. Ranges from `0` to `1` with `0.1` precision.

---

float **wind_attenuation_factor** = `0.0`

- void **set_wind_attenuation_factor**(value: float)
- float **get_wind_attenuation_factor**()

The exponential rate at which wind force decreases with distance from its origin.

Note: This wind force only applies to SoftBody3D nodes. Other physics bodies are currently not affected by wind.

---

float **wind_force_magnitude** = `0.0`

- void **set_wind_force_magnitude**(value: float)
- float **get_wind_force_magnitude**()

The magnitude of area-specific wind force.

Note: This wind force only applies to SoftBody3D nodes. Other physics bodies are currently not affected by wind.

NodePath **wind_source_path** = `NodePath("")`

- void **set_wind_source_path**(value: NodePath)
- NodePath **get_wind_source_path**()

The Node3D which is used to specify the direction and origin of an area-specific wind force. The direction is opposite to the z-axis of the Node3D's local transform, and its origin is the origin of the Node3D's local transform.

Note: This wind force only applies to SoftBody3D nodes. Other physics bodies are currently not affected by wind.

Method Descriptions

Array[Area3D] **get_overlapping_areas**() const

Returns a list of intersecting **Area3D**s. The overlapping area's CollisionObject3D.collision_layer must be part of this area's CollisionObject3D.collision_mask in order to be detected.

For performance reasons (collisions are all processed at the same time) this list is modified once during the physics step, not immediately after objects are moved. Consider using signals instead.

Array[Node3D] **get_overlapping_bodies**() const

Returns a list of intersecting PhysicsBody3Ds and GridMaps. The overlapping body's CollisionObject3D.collision_layer must be part of this area's CollisionObject3D.collision_mask in order to be detected.

For performance reasons (collisions are all processed at the same time) this list is modified once during the physics step, not immediately after objects are moved. Consider using signals instead.

---

bool **has_overlapping_areas**() const

Returns `true` if intersecting any **Area3D**s, otherwise returns `false`. The overlapping area's CollisionObject3D.collision_layer must be part of this area's CollisionObject3D.collision_mask in order to be detected.

For performance reasons (collisions are all processed at the same time) the list of overlapping areas is modified once during the physics step, not immediately after objects are moved. Consider using signals instead.

---

bool **has_overlapping_bodies**() const

Returns `true` if intersecting any PhysicsBody3Ds or GridMaps, otherwise returns `false`. The overlapping body's CollisionObject3D.collision_layer must be part of this area's CollisionObject3D.collision_mask in order to be detected.

For performance reasons (collisions are all processed at the same time) the list of overlapping bodies is modified once during the physics step, not immediately after objects are moved. Consider using signals instead.

---

bool **overlaps_area**(area: Node) const

Returns `true` if the given **Area3D** intersects or overlaps this **Area3D**, `false` otherwise.

Note: The result of this test is not immediate after moving objects. For performance, list of overlaps is updated once per frame and before the physics step. Consider using signals instead.

---

bool **overlaps_body**(body: Node) const

Returns `true` if the given physics body intersects or overlaps this **Area3D**, `false` otherwise.

Note: The result of this test is not immediate after moving objects. For performance, list of overlaps is updated once per frame and before the physics step. Consider using signals instead.

The `body` argument can either be a PhysicsBody3D or a GridMap instance. While GridMaps are not physics body themselves, they register their tiles with collision shapes as a virtual physics body.

AspectRatioContainer

Inherits: Container **<** Control **<** CanvasItem **<** Node **<** Object

A container that preserves the proportions of its child controls.

Description

A container type that arranges its child controls in a way that preserves their proportions automatically when the container is resized. Useful when a container has a dynamic size and the child nodes must adjust their sizes accordingly without losing their aspect ratios.

Tutorials

- Using Containers

Properties

AlignmentMode	alignment_horizontal	`1`
AlignmentMode	alignment_vertical	`1`
float	ratio	`1.0`
StretchMode	stretch_mode	`2`

Enumerations

enum **StretchMode**: 🔗

StretchMode **STRETCH_WIDTH_CONTROLS_HEIGHT** = `0`

The height of child controls is automatically adjusted based on the width of the container.

StretchMode **STRETCH_HEIGHT_CONTROLS_WIDTH** = `1`

The width of child controls is automatically adjusted based on the height of the container.

StretchMode **STRETCH_FIT** = `2`

The bounding rectangle of child controls is automatically adjusted to fit inside the container while keeping the aspect ratio.

StretchMode **STRETCH_COVER** = `3`

The width and height of child controls is automatically adjusted to make their bounding rectangle cover the entire area of the container while keeping the aspect ratio.

When the bounding rectangle of child controls exceed the container's size and Control.clip_contents is enabled, this allows to show only the container's area restricted by its own bounding rectangle.

---

enum **AlignmentMode**: 🔗

AlignmentMode **ALIGNMENT_BEGIN** = `0`

Aligns child controls with the beginning (left or top) of the container.

AlignmentMode **ALIGNMENT_CENTER** = `1`

Aligns child controls with the center of the container.

AlignmentMode **ALIGNMENT_END** = `2`

Aligns child controls with the end (right or bottom) of the container.

Property Descriptions

AlignmentMode **alignment_horizontal** = `1`

- void **set_alignment_horizontal**(value: AlignmentMode)
- AlignmentMode **get_alignment_horizontal**()

Specifies the horizontal relative position of child controls.

AlignmentMode **alignment_vertical** = `1`

- void **set_alignment_vertical**(value: AlignmentMode)
- AlignmentMode **get_alignment_vertical**()

Specifies the vertical relative position of child controls.

---

float **ratio** = `1.0`

- void **set_ratio**(value: float)
- float **get_ratio**()

The aspect ratio to enforce on child controls. This is the width divided by the height. The ratio depends on the stretch_mode.

AudioListener2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Overrides the location sounds are heard from.

Description

Once added to the scene tree and enabled using make_current, this node will override the location sounds are heard from. Only one **AudioListener2D** can be current. Using make_current will disable the previous **AudioListener2D**.

If there is no active **AudioListener2D** in the current Viewport, center of the screen will be used as a hearing point for the audio. **AudioListener2D** needs to be inside SceneTree to function.

Methods

void	clear_current()
bool	is_current() const
void	make_current()

Method Descriptions

void **clear_current**() 🔗

Disables the **AudioListener2D**. If it's not set as current, this method will have no effect.

---

bool **is_current**() const 🔗

Returns `true` if this **AudioListener2D** is currently active.

---

void **make_current**() 🔗

Makes the **AudioListener2D** active, setting it as the hearing point for the sounds. If there is already another active **AudioListener2D**, it will be disabled.

This method will have no effect if the **AudioListener2D** is not added to SceneTree.

AudioListener3D

Inherits: Node3D **<** Node **<** Object

Overrides the location sounds are heard from.

Description

Once added to the scene tree and enabled using make_current, this node will override the location sounds are heard from. This can be used to listen from a location different from the Camera3D.

Methods

void	clear_current()
Transform3D	get_listener_transform() const
bool	is_current() const
void	make_current()

Method Descriptions

void **clear_current**() 🔗

Disables the listener to use the current camera's listener instead.

Transform3D **get_listener_transform**() const

Returns the listener's global orthonormalized Transform3D.

bool **is_current**() const

Returns `true` if the listener was made current using make_current, `false` otherwise.

Note: There may be more than one AudioListener3D marked as "current" in the scene tree, but only the one that was made current last will be used.

void **make_current**()

Enables the listener. This will override the current camera's listener.

AudioStreamPlayer

Inherits: Node **<** Object

A node for audio playback.

Description

The **AudioStreamPlayer** node plays an audio stream non-positionally. It is ideal for user interfaces, menus, or background music.

To use this node, stream needs to be set to a valid AudioStream resource. Playing more than one sound at the same time is also supported, see max_polyphony.

If you need to play audio at a specific position, use AudioStreamPlayer2D or AudioStreamPlayer3D instead.

Tutorials

- Audio streams
- 2D Dodge The Creeps Demo [https://godotengine.org/asset-library/asset/2712]
- Audio Device Changer Demo [https://godotengine.org/asset-library/asset/2758]
- Audio Generator Demo [https://godotengine.org/asset-library/asset/2759]
- Audio Microphone Record Demo [https://godotengine.org/asset-library/asset/2760]
- Audio Spectrum Visualizer Demo [https://godotengine.org/asset-library/asset/2762]

Properties

bool	autoplay	`false`
StringName	bus	`&"Master"`
int	max_polyphony	`1`
MixTarget	mix_target	`0`
float	pitch_scale	`1.0`
PlaybackType	playback_type	`0`
bool	playing	`false`
AudioStream	stream	
bool	stream_paused	`false`
float	volume_db	`0.0`

Methods

float	get_playback_position()
AudioStreamPlayback	get_stream_playback()
bool	has_stream_playback()
void	play(from_position: float = 0.0)
void	seek(to_position: float)
void	stop()

Signals

finished()

Emitted when a sound finishes playing without interruptions. This signal is *not* emitted when calling stop, or when exiting the tree while sounds are playing.

Enumerations

enum **MixTarget**:

MixTarget **MIX_TARGET_STEREO** = 0

The audio will be played only on the first channel. This is the default.

MixTarget **MIX_TARGET_SURROUND** = `1`

The audio will be played on all surround channels.

MixTarget **MIX_TARGET_CENTER** = `2`

The audio will be played on the second channel, which is usually the center.

Property Descriptions

bool **autoplay** = `false`

- void **set_autoplay**(value: bool)
- bool **is_autoplay_enabled**()

If `true`, this node calls play when entering the tree.

StringName **bus** = `&"Master"`

- void **set_bus**(value: StringName)
- StringName **get_bus**()

The target bus name. All sounds from this node will be playing on this bus.

Note: At runtime, if no bus with the given name exists, all sounds will fall back on `"Master"`. See also AudioServer.get_bus_name.

---

int **max_polyphony** = `1`

- void **set_max_polyphony**(value: int)
- int **get_max_polyphony**()

The maximum number of sounds this node can play at the same time. Calling play after this value is reached will cut off the oldest sounds.

MixTarget **mix_target** = `0` 🔗

- void **set_mix_target**(value: MixTarget)
- MixTarget **get_mix_target**()

The mix target channels, as one of the MixTarget constants. Has no effect when two speakers or less are detected (see SpeakerMode).

---

float **pitch_scale** = `1.0` 🔗

- void **set_pitch_scale**(value: float)
- float **get_pitch_scale**()

The audio's pitch and tempo, as a multiplier of the stream's sample rate. A value of `2.0` doubles the audio's pitch, while a value of `0.5` halves the pitch.

PlaybackType **playback_type** = `0` 🔗

- void **set_playback_type**(value: PlaybackType)
- PlaybackType **get_playback_type**()

Experimental: This property may be changed or removed in future versions.

The playback type of the stream player. If set other than to the default value, it will force that playback type.

---

bool **playing** = `false`

- bool **is_playing**()

If `true`, this node is playing sounds. Setting this property has the same effect as play and stop.

AudioStream **stream**

- void **set_stream**(value: AudioStream)
- AudioStream **get_stream**()

The AudioStream resource to be played. Setting this property stops all currently playing sounds. If left empty, the **AudioStreamPlayer** does not work.

---

bool **stream_paused** = `false`

- void **set_stream_paused**(value: bool)
- bool **get_stream_paused**()

If `true`, the sounds are paused. Setting stream_paused to `false` resumes all sounds.

Note: This property is automatically changed when exiting or entering the tree, or this node is paused (see Node.process_mode).

---

float **volume_db** = `0.0`

- void **set_volume_db**(value: float)
- float **get_volume_db**()

Volume of sound, in decibel. This is an offset of the stream's volume.

Note: To convert between decibel and linear energy (like most volume sliders do), use @GlobalScope.db_to_linear and @GlobalScope.linear_to_db.

Method Descriptions

float **get_playback_position**()

Returns the position in the AudioStream of the latest sound, in seconds. Returns `0.0` if no sounds are playing.

Note: The position is not always accurate, as the AudioServer does not mix audio every processed frame. To get more accurate results, add AudioServer.get_time_since_last_mix to the returned position.

AudioStreamPlayback **get_stream_playback**()

Returns the latest AudioStreamPlayback of this node, usually the most recently created by play. If no sounds are playing, this method fails and returns an empty playback.

---

bool **has_stream_playback**()

Returns `true` if any sound is active, even if stream_paused is set to `true`. See also playing and get_stream_playback.

---

void **play**(from_position: float = 0.0)

Plays a sound from the beginning, or the given `from_position` in seconds.

---

void **seek**(to_position: float)

Restarts all sounds to be played from the given `to_position`, in seconds. Does nothing if no sounds are playing.

---

void **stop**()

Stops all sounds from this node.

AudioStreamPlayer2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Plays positional sound in 2D space.

Description

Plays audio that is attenuated with distance to the listener.

By default, audio is heard from the screen center. This can be changed by adding an AudioListener2D node to the scene and enabling it by calling AudioListener2D.make_current on it.

See also AudioStreamPlayer to play a sound non-positionally.

Note: Hiding an **AudioStreamPlayer2D** node does not disable its audio output. To temporarily disable an **AudioStreamPlayer2D**'s audio output, set volume_db to a very low value like `-100` (which isn't audible to human hearing).

Tutorials

- Audio streams

Properties

int	area_mask	`1`

float	attenuation	`1.0`
bool	autoplay	`false`
StringName	bus	`&"Master"`
float	max_distance	`2000.0`
int	max_polyphony	`1`
float	panning_strength	`1.0`
float	pitch_scale	`1.0`
PlaybackType	playback_type	`0`
bool	playing	`false`
AudioStream	stream	
bool	stream_paused	`false`
float	volume_db	`0.0`

Methods

float	get_playback_position()
AudioStreamPlayback	get_stream_playback()
bool	has_stream_playback()
void	play(from_position: float = 0.0)
void	seek(to_position: float)
void	stop()

Signals

finished() 🔗

Emitted when the audio stops playing.

Property Descriptions

int **area_mask** = `1` 🔗

- void **set_area_mask**(value: int)

- int **get_area_mask**()

Determines which Area2D layers affect the sound for reverb and audio bus effects. Areas can be used to redirect AudioStreams so that they play in a certain audio bus. An example of how you might use this is making a "water" area so that sounds played in the water are redirected through an audio bus to make them sound like they are being played underwater.

---

float **attenuation** = `1.0`

- void **set_attenuation**(value: float)
- float **get_attenuation**()

The volume is attenuated over distance with this as an exponent.

---

bool **autoplay** = `false`

- void **set_autoplay**(value: bool)
- bool **is_autoplay_enabled**()

If `true`, audio plays when added to scene tree.

StringName **bus** = `&"Master"`

- void **set_bus**(value: StringName)
- StringName **get_bus**()

Bus on which this audio is playing.

Note: When setting this property, keep in mind that no validation is performed to see if the given name matches an existing bus. This is because audio bus layouts might be loaded after this property is set.

If this given name can't be resolved at runtime, it will fall back to `"Master"`.

---

float **max_distance** = `2000.0` 🔗

- void **set_max_distance**(value: float)
- float **get_max_distance**()

Maximum distance from which audio is still hearable.

---

int **max_polyphony** = `1` 🔗

- void **set_max_polyphony**(value: int)
- int **get_max_polyphony**()

The maximum number of sounds this node can play at the same time. Playing additional sounds after this value is reached will cut off the oldest sounds.

---

float **panning_strength** = `1.0` 🔗

- void **set_panning_strength**(value: float)
- float **get_panning_strength**()

Scales the panning strength for this node by multiplying the base ProjectSettings.audio/general/2d_panning_strength with this factor. Higher values will pan audio from left to right more dramatically than lower values.

---

float **pitch_scale** = `1.0` 🔗

- void **set_pitch_scale**(value: float)
- float **get_pitch_scale**()

The pitch and the tempo of the audio, as a multiplier of the audio sample's sample rate.

PlaybackType **playback_type** = `0`

- void **set_playback_type**(value: PlaybackType)
- PlaybackType **get_playback_type**()

Experimental: This property may be changed or removed in future versions.

The playback type of the stream player. If set other than to the default value, it will force that playback type.

bool **playing** = `false`

- bool **is_playing**()

If `true`, audio is playing or is queued to be played (see play).

AudioStream **stream**

- void **set_stream**(value: AudioStream)
- AudioStream **get_stream**()

The AudioStream object to be played.

bool **stream_paused** = `false`

- void **set_stream_paused**(value: bool)
- bool **get_stream_paused**()

If `true`, the playback is paused. You can resume it by setting stream_paused to `false`.

---

float **volume_db** = `0.0` 🔗

- void **set_volume_db**(value: float)
- float **get_volume_db**()

Base volume before attenuation.

Method Descriptions

float **get_playback_position**() 🔗

Returns the position in the AudioStream.

AudioStreamPlayback **get_stream_playback**() 🔗

Returns the AudioStreamPlayback object associated with this **AudioStreamPlayer2D**.

---

bool **has_stream_playback**() 🔗

Returns whether the AudioStreamPlayer can return the AudioStreamPlayback object or not.

---

void **play**(from_position: float = 0.0) 🔗

Queues the audio to play on the next physics frame, from the given position `from_position`, in seconds.

---

void **seek**(to_position: float)

Sets the position from which audio will be played, in seconds.

void **stop**()

Stops the audio.

AudioStreamPlayer3D

Inherits: Node3D **<** Node **<** Object

Plays positional sound in 3D space.

Description

Plays audio with positional sound effects, based on the relative position of the audio listener. Positional effects include distance attenuation, directionality, and the Doppler effect. For greater realism, a low-pass filter is applied to distant sounds. This can be disabled by setting attenuation_filter_cutoff_hz to `20500`.

By default, audio is heard from the camera position. This can be changed by adding an AudioListener3D node to the scene and enabling it by calling AudioListener3D.make_current on it.

See also AudioStreamPlayer to play a sound non-positionally.

Note: Hiding an **AudioStreamPlayer3D** node does not disable its audio output. To temporarily disable an **AudioStreamPlayer3D**'s audio output, set volume_db to a very low value like `-100` (which isn't audible to human hearing).

Tutorials

- Audio streams

Properties

int	area_mask	`1`
float	attenuation_filter_cutoff_hz	`5000.0`
float	attenuation_filter_db	`-24.0`
AttenuationModel	attenuation_model	`0`
bool	autoplay	`false`
StringName	bus	`&"Master"`
DopplerTracking	doppler_tracking	`0`
float	emission_angle_degrees	`45.0`
bool	emission_angle_enabled	`false`
float	emission_angle_filter_attenuation_db	`-12.0`
float	max_db	`3.0`
float	max_distance	`0.0`
int	max_polyphony	`1`

float	panning_strength	`1.0`
float	pitch_scale	`1.0`
PlaybackType	playback_type	`0`
bool	playing	`false`
AudioStream	stream	
bool	stream_paused	`false`
float	unit_size	`10.0`
float	volume_db	`0.0`

Methods

float	get_playback_position()
AudioStreamPlayback	get_stream_playback()
bool	has_stream_playback()

void	play(from_position: float = 0.0)
void	seek(to_position: float)
void	stop()

Signals

finished()

Emitted when the audio stops playing.

Enumerations

enum **AttenuationModel**:

AttenuationModel **ATTENUATION_INVERSE_DISTANCE** = `0`

Attenuation of loudness according to linear distance.

AttenuationModel **ATTENUATION_INVERSE_SQUARE_DISTANCE** = `1`

Attenuation of loudness according to squared distance.

AttenuationModel **ATTENUATION_LOGARITHMIC** = `2`

Attenuation of loudness according to logarithmic distance.

AttenuationModel **ATTENUATION_DISABLED** = `3`

No attenuation of loudness according to distance. The sound will still be heard positionally, unlike an AudioStreamPlayer. ATTENUATION_DISABLED can be combined with a max_distance value greater than `0.0` to achieve linear attenuation clamped to a sphere of a defined size.

---

enum **DopplerTracking**: 🔗

DopplerTracking **DOPPLER_TRACKING_DISABLED** = `0`

Disables doppler tracking.

DopplerTracking **DOPPLER_TRACKING_IDLE_STEP** = `1`

Executes doppler tracking during process frames (see Node.NOTIFICATION_INTERNAL_PROCESS).

DopplerTracking **DOPPLER_TRACKING_PHYSICS_STEP** = `2`

Executes doppler tracking during physics frames (see Node.NOTIFICATION_INTERNAL_PHYSICS_PROCESS).

Property Descriptions

int **area_mask** = `1` 🔗

- void **set_area_mask**(value: int)
- int **get_area_mask**()

Determines which Area3D layers affect the sound for reverb and audio bus effects. Areas can be used to redirect AudioStreams so that they play in a certain audio bus. An example of how you might use this is making a "water" area so that sounds played in the water are redirected through an audio bus to make them sound like they are being played underwater.

---

float **attenuation_filter_cutoff_hz** = `5000.0`

- void **set_attenuation_filter_cutoff_hz**(value: float)
- float **get_attenuation_filter_cutoff_hz**()

The cutoff frequency of the attenuation low-pass filter, in Hz. A sound above this frequency is attenuated more than a sound below this frequency. To disable this effect, set this to `20500` as this frequency is above the human hearing limit.

---

float **attenuation_filter_db** = `-24.0`

- void **set_attenuation_filter_db**(value: float)
- float **get_attenuation_filter_db**()

Amount how much the filter affects the loudness, in decibels.

AttenuationModel **attenuation_model** = `0`

- void **set_attenuation_model**(value: AttenuationModel)
- AttenuationModel **get_attenuation_model**()

Decides if audio should get quieter with distance linearly, quadratically, logarithmically, or not be affected by distance, effectively disabling attenuation.

---

bool **autoplay** = `false`

- void **set_autoplay**(value: bool)

- bool **is_autoplay_enabled**()

If `true`, audio plays when the AudioStreamPlayer3D node is added to scene tree.

StringName **bus** = `&"Master"` 🔗

- void **set_bus**(value: StringName)
- StringName **get_bus**()

The bus on which this audio is playing.

Note: When setting this property, keep in mind that no validation is performed to see if the given name matches an existing bus. This is because audio bus layouts might be loaded after this property is set. If this given name can't be resolved at runtime, it will fall back to `"Master"`.

DopplerTracking **doppler_tracking** = `0` 🔗

- void **set_doppler_tracking**(value: DopplerTracking)
- DopplerTracking **get_doppler_tracking**()

Decides in which step the Doppler effect should be calculated.

---

float **emission_angle_degrees** = `45.0` 🔗

- void **set_emission_angle**(value: float)
- float **get_emission_angle**()

The angle in which the audio reaches a listener unattenuated.

---

bool **emission_angle_enabled** = `false` 🔗

- void **set_emission_angle_enabled**(value: bool)
- bool **is_emission_angle_enabled**()

If `true`, the audio should be attenuated according to the direction of the sound.

---

float **emission_angle_filter_attenuation_db** = `-12.0` 🔗

- void **set_emission_angle_filter_attenuation_db**(value: float)
- float **get_emission_angle_filter_attenuation_db**()

Attenuation factor used if listener is outside of emission_angle_degrees and emission_angle_enabled is set, in decibels.

---

float **max_db** = `3.0` 🔗

- void **set_max_db**(value: float)
- float **get_max_db**()

Sets the absolute maximum of the sound level, in decibels.

---

float **max_distance** = `0.0` 🔗

- void **set_max_distance**(value: float)
- float **get_max_distance**()

The distance past which the sound can no longer be heard at all. Only has an effect if set to a value greater than `0.0`. max_distance works in tandem with unit_size. However, unlike unit_size whose behavior depends on the attenuation_model, max_distance always works in a linear fashion. This can be used to prevent the

AudioStreamPlayer3D from requiring audio mixing when the listener is far away, which saves CPU resources.

---

int **max_polyphony** = `1`

- void **set_max_polyphony**(value: int)
- int **get_max_polyphony**()

The maximum number of sounds this node can play at the same time. Playing additional sounds after this value is reached will cut off the oldest sounds.

---

float **panning_strength** = `1.0`

- void **set_panning_strength**(value: float)
- float **get_panning_strength**()

Scales the panning strength for this node by multiplying the base ProjectSettings.audio/general/3d_panning_strength with this factor. Higher values will pan audio from left to right more dramatically than lower values.

---

float **pitch_scale** = `1.0`

- void **set_pitch_scale**(value: float)
- float **get_pitch_scale**()

The pitch and the tempo of the audio, as a multiplier of the audio sample's sample rate.

PlaybackType **playback_type** = `0`

- void **set_playback_type**(value: PlaybackType)

- PlaybackType **get_playback_type**()

Experimental: This property may be changed or removed in future versions.

The playback type of the stream player. If set other than to the default value, it will force that playback type.

bool **playing** = `false` 🔗

- bool **is_playing**()

If `true`, audio is playing or is queued to be played (see play).

AudioStream **stream** 🔗

- void **set_stream**(value: AudioStream)
- AudioStream **get_stream**()

The AudioStream resource to be played.

bool **stream_paused** = `false` 🔗

- void **set_stream_paused**(value: bool)
- bool **get_stream_paused**()

If `true`, the playback is paused. You can resume it by setting stream_paused to `false`.

float **unit_size** = `10.0` 🔗

- void **set_unit_size**(value: float)
- float **get_unit_size**()

The factor for the attenuation effect. Higher values make the sound audible over a larger distance.

---

float **volume_db** = `0.0`

- void **set_volume_db**(value: float)
- float **get_volume_db**()

The base sound level before attenuation, in decibels.

Method Descriptions

float **get_playback_position**()

Returns the position in the AudioStream.

AudioStreamPlayback **get_stream_playback**()

Returns the AudioStreamPlayback object associated with this **AudioStreamPlayer3D**.

---

bool **has_stream_playback**()

Returns whether the AudioStreamPlayer can return the AudioStreamPlayback object or not.

---

void **play**(from_position: float = 0.0)

Queues the audio to play on the next physics frame, from the given position `from_position`, in seconds.

---

void **seek**(to_position: float) 🔗

Sets the position from which audio will be played, in seconds.

---

void **stop**() 🔗

Stops the audio.

BackBufferCopy

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A node that copies a region of the screen to a buffer for access in shader code.

Description

Node for back-buffering the currently-displayed screen. The region defined in the **BackBufferCopy** node is buffered with the content of the screen it covers, or the entire screen according to the copy_mode. It can be accessed in shader scripts using the screen texture (i.e. a uniform sampler with `hint_screen_texture`).

Note: Since this node inherits from Node2D (and not Control), anchors and margins won't apply to child Control-derived nodes. This can be problematic when resizing the window. To avoid this, add Control-derived nodes as *siblings* to the **BackBufferCopy** node instead of adding them as children.

Properties

CopyMode	copy_mode	`1`
Rect2	rect	`Rect2(-100, -100, 200, 200)`

Enumerations

enum **CopyMode**: 🔗

CopyMode **COPY_MODE_DISABLED** = `0`

Disables the buffering mode. This means the **BackBufferCopy** node will directly use the portion of screen it covers.

CopyMode **COPY_MODE_RECT** = `1`

BackBufferCopy buffers a rectangular region.

CopyMode **COPY_MODE_VIEWPORT** = `2`

BackBufferCopy buffers the entire screen.

Property Descriptions

CopyMode **copy_mode** = `1` 🔗

- void **set_copy_mode**(value: CopyMode)
- CopyMode **get_copy_mode**()

Buffer mode. See CopyMode constants.

Rect2 **rect** = `Rect2(-100, -100, 200, 200)` 🔗

- void **set_rect**(value: Rect2)
- Rect2 **get_rect**()

The area covered by the **BackBufferCopy**. Only used if copy_mode is COPY_MODE_RECT.

BaseButton

Inherits: Control **<** CanvasItem **<** Node **<** Object

Inherited By: Button, LinkButton, TextureButton

Abstract base class for GUI buttons.

Description

BaseButton is an abstract base class for GUI buttons. It doesn't display anything by itself.

Properties

ActionMode	action_mode	`1`
ButtonGroup	button_group	
BitField[MouseButtonMask]	button_mask	`1`
bool	button_pressed	`false`
bool	disabled	`false`
FocusMode	focus_mode	`2` (overrides Control)

bool	keep_pressed_outside	`false`
Shortcut	shortcut	
bool	shortcut_feedback	`true`
bool	shortcut_in_tooltip	`true`
bool	toggle_mode	`false`

Methods

void	_pressed() virtual
void	_toggled(toggled_on: bool) virtual
DrawMode	get_draw_mode() const
bool	is_hovered() const
void	set_pressed_no_signal(pressed: bool)

Signals

button_down() 🔗

Emitted when the button starts being held down.

button_up() 🔗

Emitted when the button stops being held down.

pressed() 🔗

Emitted when the button is toggled or pressed. This is on button_down if action_mode is ACTION_MODE_BUTTON_PRESS and on button_up otherwise.

If you need to know the button's pressed state (and toggle_mode is active), use toggled instead.

toggled(toggled_on: bool) 🔗

Emitted when the button was just toggled between pressed and normal states (only if toggle_mode is active). The new state is contained in the `toggled_on` argument.

Enumerations

enum **DrawMode**: 🔗

DrawMode **DRAW_NORMAL** = `0`

The normal state (i.e. not pressed, not hovered, not toggled and enabled) of buttons.

DrawMode **DRAW_PRESSED** = `1`

The state of buttons are pressed.

DrawMode **DRAW_HOVER** = `2`

The state of buttons are hovered.

DrawMode **DRAW_DISABLED** = `3`

The state of buttons are disabled.

DrawMode **DRAW_HOVER_PRESSED** = `4`

The state of buttons are both hovered and pressed.

---

enum **ActionMode**:

ActionMode **ACTION_MODE_BUTTON_PRESS** = `0`

Require just a press to consider the button clicked.

ActionMode **ACTION_MODE_BUTTON_RELEASE** = `1`

Require a press and a subsequent release before considering the button clicked.

Property Descriptions

ActionMode **action_mode** = `1`

- void **set_action_mode**(value: ActionMode)

- ActionMode **get_action_mode**()

Determines when the button is considered clicked, one of the ActionMode constants.

ButtonGroup **button_group**

- void **set_button_group**(value: ButtonGroup)
- ButtonGroup **get_button_group**()

The ButtonGroup associated with the button. Not to be confused with node groups.

Note: The button will be configured as a radio button if a ButtonGroup is assigned to it.

BitField[MouseButtonMask] **button_mask** = `1`

- void **set_button_mask**(value: BitField[MouseButtonMask])
- BitField[MouseButtonMask] **get_button_mask**()

Binary mask to choose which mouse buttons this button will respond to.

To allow both left-click and right-click, use `MOUSE_BUTTON_MASK_LEFT | MOUSE_BUTTON_MASK_RIGHT`.

---

bool **button_pressed** = `false`

- void **set_pressed**(value: bool)
- bool **is_pressed**()

If `true`, the button's state is pressed. Means the button is pressed down or toggled (if toggle_mode is active). Only works if toggle_mode is `true`.

Note: Setting button_pressed will result in toggled to be emitted. If you want to change the pressed state without emitting that signal, use set_pressed_no_signal.

---

bool **disabled** = `false`

- void **set_disabled**(value: bool)
- bool **is_disabled**()

If `true`, the button is in disabled state and can't be clicked or toggled.

---

bool **keep_pressed_outside** = `false`

- void **set_keep_pressed_outside**(value: bool)
- bool **is_keep_pressed_outside**()

If `true`, the button stays pressed when moving the cursor outside the button while pressing it.

Note: This property only affects the button's visual appearance. Signals will be emitted at the same moment regardless of this property's value.

Shortcut **shortcut**

- void **set_shortcut**(value: Shortcut)
- Shortcut **get_shortcut**()

Shortcut associated to the button.

---

bool **shortcut_feedback** = `true`

- void **set_shortcut_feedback**(value: bool)
- bool **is_shortcut_feedback**()

If `true`, the button will highlight for a short amount of time when its shortcut is activated. If `false` and toggle_mode is `false`, the shortcut will activate without any visual feedback.

---

bool **shortcut_in_tooltip** = `true`

- void **set_shortcut_in_tooltip**(value: bool)
- bool **is_shortcut_in_tooltip_enabled**()

If `true`, the button will add information about its shortcut in the tooltip.

---

bool **toggle_mode** = `false`

- void **set_toggle_mode**(value: bool)
- bool **is_toggle_mode**()

If `true`, the button is in toggle mode. Makes the button flip state between pressed and unpressed each time its area is clicked.

Method Descriptions

void **_pressed**() virtual

Called when the button is pressed. If you need to know the button's pressed state (and toggle_mode is active), use _toggled instead.

---

void **_toggled**(toggled_on: bool) virtual

Called when the button is toggled (only if toggle_mode is active).

DrawMode **get_draw_mode**() const

Returns the visual state used to draw the button. This is useful mainly when implementing your own draw code by either overriding _draw() or connecting to "draw" signal. The visual state of the button is defined by the DrawMode enum.

---

bool **is_hovered**() const

Returns `true` if the mouse has entered the button and has not left it yet.

---

void **set_pressed_no_signal**(pressed: bool)

Changes the button_pressed state of the button, without emitting toggled. Use when you just want to change the state of the button without sending the pressed event (e.g. when initializing scene). Only works if toggle_mode is `true`.

Note: This method doesn't unpress other buttons in button_group.

Bone2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A joint used with Skeleton2D to control and animate other nodes.

Description

A hierarchy of **Bone2D**s can be bound to a Skeleton2D to control and animate other Node2D nodes.

You can use **Bone2D** and Skeleton2D nodes to animate 2D meshes created with the Polygon2D UV editor.

Each bone has a rest transform that you can reset to with apply_rest. These rest poses are relative to the bone's parent.

If in the editor, you can set the rest pose of an entire skeleton using a menu option, from the code, you need to iterate over the bones to set their individual rest poses.

Properties

Transform2D	rest	`Transform2D(0, 0, 0, 0, 0, 0)`

Methods

void	apply_rest()

bool	get_autocalculate_length_and_angle() const
float	get_bone_angle() const
int	get_index_in_skeleton() const
float	get_length() const
Transform2D	get_skeleton_rest() const
void	set_autocalculate_length_and_angle(auto_calculate: bool)
void	set_bone_angle(angle: float)
void	set_length(length: float)

Property Descriptions

Transform2D **rest** = `Transform2D(0, 0, 0, 0, 0, 0)` 🔗

- void **set_rest**(value: Transform2D)
- Transform2D **get_rest**()

Rest transform of the bone. You can reset the node's transforms to this value using apply_rest.

Method Descriptions

void **apply_rest**()

Resets the bone to the rest pose. This is equivalent to setting Node2D.transform to rest.

bool **get_autocalculate_length_and_angle**() const

Returns whether this **Bone2D** is going to autocalculate its length and bone angle using its first **Bone2D** child node, if one exists. If there are no **Bone2D** children, then it cannot autocalculate these values and will print a warning.

float **get_bone_angle**() const

Returns the angle of the bone in the **Bone2D**.

Note: This is different from the **Bone2D**'s rotation. The bone's angle is the rotation of the bone shown by the gizmo, which is unaffected by the **Bone2D**'s Node2D.transform.

int **get_index_in_skeleton**() const

Returns the node's index as part of the entire skeleton. See Skeleton2D.

float **get_length**() const

Returns the length of the bone in the **Bone2D** node.

Transform2D **get_skeleton_rest**() const

Returns the node's rest Transform2D if it doesn't have a parent, or its rest pose relative to its parent.

---

void **set_autocalculate_length_and_angle**(auto_calculate: bool)

When set to `true`, the **Bone2D** node will attempt to automatically calculate the bone angle and length using the first child **Bone2D** node, if one exists. If none exist, the **Bone2D** cannot automatically calculate these values and will print a warning.

---

void **set_bone_angle**(angle: float)

Sets the bone angle for the **Bone2D**. This is typically set to the rotation from the **Bone2D** to a child **Bone2D** node.

Note: This is different from the **Bone2D**'s rotation. The bone's angle is the rotation of the bone shown by the gizmo, which is unaffected by the **Bone2D**'s Node2D.transform.

---

void **set_length**(length: float)

Sets the length of the bone in the **Bone2D**.

BoneAttachment3D

Inherits: Node3D **<** Node **<** Object

A node that dynamically copies or overrides the 3D transform of a bone in its parent Skeleton3D.

Description

This node selects a bone in a Skeleton3D and attaches to it. This means that the **BoneAttachment3D** node will either dynamically copy or override the 3D transform of the selected bone.

Properties

int	bone_idx	`-1`
String	bone_name	`""`
bool	override_pose	`false`

Methods

NodePath	get_external_skeleton() const

bool	get_use_external_skeleton() const
void	on_skeleton_update()
void	set_external_skeleton(external_skeleton: NodePath)
void	set_use_external_skeleton(use_external_skeleton: bool)

Property Descriptions

int **bone_idx** = `-1` 🔗

- void **set_bone_idx**(value: int)
- int **get_bone_idx**()

The index of the attached bone.

String **bone_name** = `""` 🔗

- void **set_bone_name**(value: String)
- String **get_bone_name**()

The name of the attached bone.

---

bool **override_pose** = `false` 🔗

- void **set_override_pose**(value: bool)
- bool **get_override_pose**()

Whether the BoneAttachment3D node will override the bone pose of the bone it is attached to. When set to `true`, the BoneAttachment3D node can change the pose of the bone. When set to `false`, the BoneAttachment3D will always be set to the bone's transform.

Note: This override performs interruptively in the skeleton update process using signals due to the old design. It may cause unintended behavior when used at the same time with SkeletonModifier3D.

Method Descriptions

NodePath **get_external_skeleton**() const

Returns the NodePath to the external Skeleton3D node, if one has been set.

---

bool **get_use_external_skeleton**() const

Returns whether the BoneAttachment3D node is using an external Skeleton3D rather than attempting to use its parent node as the Skeleton3D.

---

void **on_skeleton_update**()

A function that is called automatically when the Skeleton3D is updated. This function is where the **BoneAttachment3D** node updates its position so it is correctly bound when it is *not* set to override the bone pose.

---

void **set_external_skeleton**(external_skeleton: NodePath)

Sets the NodePath to the external skeleton that the BoneAttachment3D node should use. See set_use_external_skeleton to enable the external Skeleton3D node.

void **set_use_external_skeleton**(use_external_skeleton: bool) 🔗

Sets whether the BoneAttachment3D node will use an external Skeleton3D node rather than attempting to use its parent node as the Skeleton3D. When set to `true`, the BoneAttachment3D node will use the external Skeleton3D node set in set_external_skeleton.

BoxContainer

Inherits: Container **<** Control **<** CanvasItem **<** Node **<** Object

Inherited By: HBoxContainer, VBoxContainer

A container that arranges its child controls horizontally or vertically.

Description

A container that arranges its child controls horizontally or vertically, rearranging them automatically when their minimum size changes.

Tutorials

- Using Containers

Properties

AlignmentMode	alignment	`0`
bool	vertical	`false`

Methods

Control	add_spacer(begin: bool)

Theme Properties

int	separation	`4`

Enumerations

enum **AlignmentMode**:

AlignmentMode **ALIGNMENT_BEGIN** = `0`

The child controls will be arranged at the beginning of the container, i.e. top if orientation is vertical, left if orientation is horizontal (right for RTL layout).

AlignmentMode **ALIGNMENT_CENTER** = `1`

The child controls will be centered in the container.

AlignmentMode **ALIGNMENT_END** = `2`

The child controls will be arranged at the end of the container, i.e. bottom if orientation is vertical, right if orientation is horizontal (left for RTL layout).

Property Descriptions

AlignmentMode **alignment** = `0`

- void **set_alignment**(value: AlignmentMode)

- AlignmentMode **get_alignment**()

The alignment of the container's children (must be one of ALIGNMENT_BEGIN, ALIGNMENT_CENTER, or ALIGNMENT_END).

---

bool **vertical** = `false`

- void **set_vertical**(value: bool)
- bool **is_vertical**()

If `true`, the **BoxContainer** will arrange its children vertically, rather than horizontally.

Can't be changed when using HBoxContainer and VBoxContainer.

Method Descriptions

Control **add_spacer**(begin: bool)

Adds a Control node to the box as a spacer. If `begin` is `true`, it will insert the Control node in front of all other children.

Theme Property Descriptions

int **separation** = `4`

The space between the **BoxContainer**'s elements, in pixels.

Button

Inherits: BaseButton **<** Control **<** CanvasItem **<** Node **<** Object

Inherited By: CheckBox, CheckButton, ColorPickerButton, MenuButton, OptionButton

A themed button that can contain text and an icon.

Description

Button is the standard themed button. It can contain text and an icon, and it will display them according to the current Theme.

Example of creating a button and assigning an action when pressed by code:

GDScript

```
func _ready():
    var button = Button.new()
    button.text = "Click me"
    button.pressed.connect(self._button_pressed)
    add_child(button)

func _button_pressed():
    print("Hello world!")
```

C#

```
public override void _Ready()
{
    var button = new Button();
    button.Text = "Click me";
    button.Pressed += ButtonPressed;
    AddChild(button);
}

private void ButtonPressed()
{
    GD.Print("Hello world!");
}
```

See also BaseButton which contains common properties and methods associated with this node.

Note: Buttons do not interpret touch input and therefore don't support multitouch, since mouse emulation can only press one button at a given time. Use TouchScreenButton for buttons that trigger gameplay movement or actions.

Tutorials

- 2D Dodge The Creeps Demo [https://godotengine.org/asset-library/asset/2712]
- Operating System Testing Demo [https://godotengine.org/asset-library/asset/2789]

Properties

HorizontalAlignment	alignment	`1`
AutowrapMode	autowrap_mode	`0`
bool	clip_text	`false`
bool	expand_icon	`false`

bool	flat	`false`
Texture2D	icon	
HorizontalAlignment	icon_alignment	`0`
String	language	`""`
String	text	`""`
TextDirection	text_direction	`0`
OverrunBehavior	text_overrun_behavior	`0`
VerticalAlignment	vertical_icon_alignment	`1`

Theme Properties

Color	font_color	`Color(0.875, 0.875, 0.875, 1)`
Color	font_disabled_color	`Color(0.875, 0.875, 0.875, 0.5)`

Color	font_focus_color	`Color(0.95, 0.95, 0.95, 1)`
Color	font_hover_color	`Color(0.95, 0.95, 0.95, 1)`
Color	font_hover_pressed_color	`Color(1, 1, 1, 1)`
Color	font_outline_color	`Color(0, 0, 0, 1)`
Color	font_pressed_color	`Color(1, 1, 1, 1)`
Color	icon_disabled_color	`Color(1, 1, 1, 0.4)`
Color	icon_focus_color	`Color(1, 1, 1, 1)`
Color	icon_hover_color	`Color(1, 1, 1, 1)`
Color	icon_hover_pressed_color	`Color(1, 1, 1, 1)`
Color	icon_normal_color	`Color(1, 1, 1, 1)`
Color	icon_pressed_color	`Color(1, 1, 1, 1)`
int	align_to_largest_stylebox	`0`
int	h_separation	`4`

int	icon_max_width	0
int	outline_size	0
Font	font	
int	font_size	
Texture2D	icon	
StyleBox	disabled	
StyleBox	disabled_mirrored	
StyleBox	focus	
StyleBox	hover	
StyleBox	hover_mirrored	
StyleBox	hover_pressed	
StyleBox	hover_pressed_mirrored	

StyleBox	normal
StyleBox	normal_mirrored
StyleBox	pressed
StyleBox	pressed_mirrored

Property Descriptions

HorizontalAlignment **alignment** = `1` 🔗

- void **set_text_alignment**(value: HorizontalAlignment)
- HorizontalAlignment **get_text_alignment**()

Text alignment policy for the button's text, use one of the HorizontalAlignment constants.

AutowrapMode **autowrap_mode** = `0` 🔗

- void **set_autowrap_mode**(value: AutowrapMode)
- AutowrapMode **get_autowrap_mode**()

If set to something other than TextServer.AUTOWRAP_OFF, the text gets wrapped inside the node's bounding rectangle.

---

bool **clip_text** = `false` 🔗

- void **set_clip_text**(value: bool)

- bool **get_clip_text**()

When this property is enabled, text that is too large to fit the button is clipped, when disabled the Button will always be wide enough to hold the text.

bool **expand_icon** = `false` 🔗

- void **set_expand_icon**(value: bool)
- bool **is_expand_icon**()

When enabled, the button's icon will expand/shrink to fit the button's size while keeping its aspect. See also icon_max_width.

bool **flat** = `false` 🔗

- void **set_flat**(value: bool)
- bool **is_flat**()

Flat buttons don't display decoration.

Texture2D **icon** 🔗

- void **set_button_icon**(value: Texture2D)
- Texture2D **get_button_icon**()

Button's icon, if text is present the icon will be placed before the text.

To edit margin and spacing of the icon, use h_separation theme property and `content_margin_*` properties of the used StyleBoxes.

HorizontalAlignment **icon_alignment** = `0` 🔗

- void **set_icon_alignment**(value: HorizontalAlignment)
- HorizontalAlignment **get_icon_alignment**()

Specifies if the icon should be aligned horizontally to the left, right, or center of a button. Uses the same HorizontalAlignment constants as the text alignment. If centered horizontally and vertically, text will draw on top of the icon.

String **language** = `""` 🔗

- void **set_language**(value: String)
- String **get_language**()

Language code used for line-breaking and text shaping algorithms, if left empty current locale is used instead.

String **text** = `""` 🔗

- void **set_text**(value: String)
- String **get_text**()

The button's text that will be displayed inside the button's area.

TextDirection **text_direction** = `0` 🔗

- void **set_text_direction**(value: TextDirection)
- TextDirection **get_text_direction**()

Base text writing direction.

OverrunBehavior **text_overrun_behavior** = `0` 🔗

- void **set_text_overrun_behavior**(value: OverrunBehavior)

- OverrunBehavior **get_text_overrun_behavior**()

Sets the clipping behavior when the text exceeds the node's bounding rectangle. See OverrunBehavior for a description of all modes.

VerticalAlignment **vertical_icon_alignment** = `1`

- void **set_vertical_icon_alignment**(value: VerticalAlignment)
- VerticalAlignment **get_vertical_icon_alignment**()

Specifies if the icon should be aligned vertically to the top, bottom, or center of a button. Uses the same VerticalAlignment constants as the text alignment. If centered horizontally and vertically, text will draw on top of the icon.

Theme Property Descriptions

Color **font_color** = `Color(0.875, 0.875, 0.875, 1)`

Default text Color of the **Button**.

Color **font_disabled_color** = `Color(0.875, 0.875, 0.875, 0.5)`

Text Color used when the **Button** is disabled.

Color **font_focus_color** = `Color(0.95, 0.95, 0.95, 1)`

Text Color used when the **Button** is focused. Only replaces the normal text color of the button. Disabled, hovered, and pressed states take precedence over this color.

Color **font_hover_color** = `Color(0.95, 0.95, 0.95, 1)`

Text Color used when the **Button** is being hovered.

Color **font_hover_pressed_color** = `Color(1, 1, 1, 1)`

Text Color used when the **Button** is being hovered and pressed.

Color **font_outline_color** = `Color(0, 0, 0, 1)`

The tint of text outline of the **Button**.

Color **font_pressed_color** = `Color(1, 1, 1, 1)`

Text Color used when the **Button** is being pressed.

Color **icon_disabled_color** = `Color(1, 1, 1, 0.4)`

Icon modulate Color used when the **Button** is disabled.

Color **icon_focus_color** = `Color(1, 1, 1, 1)`

Icon modulate Color used when the **Button** is focused. Only replaces the normal modulate color of the button. Disabled, hovered, and pressed states take precedence over this color.

Color **icon_hover_color** = `Color(1, 1, 1, 1)`

Icon modulate Color used when the **Button** is being hovered.

Color **icon_hover_pressed_color** = `Color(1, 1, 1, 1)` 🔗

Icon modulate Color used when the **Button** is being hovered and pressed.

Color **icon_normal_color** = `Color(1, 1, 1, 1)` 🔗

Default icon modulate Color of the **Button**.

Color **icon_pressed_color** = `Color(1, 1, 1, 1)` 🔗

Icon modulate Color used when the **Button** is being pressed.

int **align_to_largest_stylebox** = `0` 🔗

This constant acts as a boolean. If `true`, the minimum size of the button and text/icon alignment is always based on the largest stylebox margins, otherwise it's based on the current button state stylebox margins.

int **h_separation** = `4` 🔗

The horizontal space between **Button**'s icon and text. Negative values will be treated as `0` when used.

int **icon_max_width** = `0` 🔗

The maximum allowed width of the **Button**'s icon. This limit is applied on top of the default size of the icon, or its expanded size if expand_icon is `true`. The height is adjusted according to the icon's ratio. If the button has additional icons (e.g. CheckBox), they will also be limited.

---

int **outline_size** = `0`

The size of the text outline.

Note: If using a font with FontFile.multichannel_signed_distance_field enabled, its FontFile.msdf_pixel_range must be set to at least *twice* the value of outline_size for outline rendering to look correct. Otherwise, the outline may appear to be cut off earlier than intended.

Font **font**

Font of the **Button**'s text.

---

int **font_size**

Font size of the **Button**'s text.

Texture2D **icon**

Default icon for the **Button**. Appears only if icon is not assigned.

StyleBox **disabled**

StyleBox used when the **Button** is disabled.

StyleBox **disabled_mirrored**

StyleBox used when the **Button** is disabled (for right-to-left layouts).

StyleBox **focus** 🔗

StyleBox used when the **Button** is focused. The focus StyleBox is displayed *over* the base StyleBox, so a partially transparent StyleBox should be used to ensure the base StyleBox remains visible. A StyleBox that represents an outline or an underline works well for this purpose. To disable the focus visual effect, assign a StyleBoxEmpty resource. Note that disabling the focus visual effect will harm keyboard/controller navigation usability, so this is not recommended for accessibility reasons.

StyleBox **hover** 🔗

StyleBox used when the **Button** is being hovered.

StyleBox **hover_mirrored** 🔗

StyleBox used when the **Button** is being hovered (for right-to-left layouts).

StyleBox **hover_pressed** 🔗

StyleBox used when the **Button** is being pressed and hovered at the same time.

StyleBox **hover_pressed_mirrored** 🔗

StyleBox used when the **Button** is being pressed and hovered at the same time (for right-to-left layouts).

StyleBox **normal**

Default StyleBox for the **Button**.

StyleBox **normal_mirrored**

Default StyleBox for the **Button** (for right-to-left layouts).

StyleBox **pressed**

StyleBox used when the **Button** is being pressed.

StyleBox **pressed_mirrored**

StyleBox used when the **Button** is being pressed (for right-to-left layouts).

Camera2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Camera node for 2D scenes.

Description

Camera node for 2D scenes. It forces the screen (current layer) to scroll following this node. This makes it easier (and faster) to program scrollable scenes than manually changing the position of CanvasItem-based nodes.

Cameras register themselves in the nearest Viewport node (when ascending the tree). Only one camera can be active per viewport. If no viewport is available ascending the tree, the camera will register in the global viewport.

This node is intended to be a simple helper to get things going quickly, but more functionality may be desired to change how the camera works. To make your own custom camera node, inherit it from Node2D and change the transform of the canvas by setting Viewport.canvas_transform in Viewport (you can obtain the current Viewport by using Node.get_viewport).

Note that the **Camera2D** node's `position` doesn't represent the actual position of the screen, which may differ due to applied smoothing or limits. You can use get_screen_center_position to get the real position.

Tutorials

- 2D Platformer Demo [https://godotengine.org/asset-library/asset/2727]
- 2D Isometric Demo [https://godotengine.org/asset-library/asset/2718]

Properties

AnchorMode	anchor_mode	`1`
Node	custom_viewport	
float	drag_bottom_margin	`0.2`
bool	drag_horizontal_enabled	`false`
float	drag_horizontal_offset	`0.0`
float	drag_left_margin	`0.2`
float	drag_right_margin	`0.2`
float	drag_top_margin	`0.2`
bool	drag_vertical_enabled	`false`
float	drag_vertical_offset	`0.0`

bool	editor_draw_drag_margin	`false`
bool	editor_draw_limits	`false`
bool	editor_draw_screen	`true`
bool	enabled	`true`
bool	ignore_rotation	`true`
int	limit_bottom	`10000000`
int	limit_left	`-10000000`
int	limit_right	`10000000`
bool	limit_smoothed	`false`
int	limit_top	`-10000000`
Vector2	offset	`Vector2(0, 0)`
bool	position_smoothing_enabled	`false`

float	position_smoothing_speed	`5.0`
Camera2DProcessCallback	process_callback	`1`
bool	rotation_smoothing_enabled	`false`
float	rotation_smoothing_speed	`5.0`
Vector2	zoom	`Vector2(1, 1)`

Methods

void	align()
void	force_update_scroll()
float	get_drag_margin(margin: Side) const
int	get_limit(margin: Side) const
Vector2	get_screen_center_position() const
Vector2	get_target_position() const

bool	is_current() const
void	make_current()
void	reset_smoothing()
void	set_drag_margin(margin: Side, drag_margin: float)
void	set_limit(margin: Side, limit: int)

Enumerations

enum **AnchorMode**: 🔗

AnchorMode **ANCHOR_MODE_FIXED_TOP_LEFT** = `0`

The camera's position is fixed so that the top-left corner is always at the origin.

AnchorMode **ANCHOR_MODE_DRAG_CENTER** = `1`

The camera's position takes into account vertical/horizontal offsets and the screen size.

---

enum **Camera2DProcessCallback**: 🔗

Camera2DProcessCallback **CAMERA2D_PROCESS_PHYSICS** = `0`

The camera updates during physics frames (see Node.NOTIFICATION_INTERNAL_PHYSICS_PROCESS).

Camera2DProcessCallback **CAMERA2D_PROCESS_IDLE** = `1`

The camera updates during process frames (see Node.NOTIFICATION_INTERNAL_PROCESS).

Property Descriptions

AnchorMode **anchor_mode** = `1` 🔗

- void **set_anchor_mode**(value: AnchorMode)
- AnchorMode **get_anchor_mode**()

The Camera2D's anchor point. See AnchorMode constants.

Node **custom_viewport** 🔗

- void **set_custom_viewport**(value: Node)
- Node **get_custom_viewport**()

The custom Viewport node attached to the **Camera2D**. If `null` or not a Viewport, uses the default viewport instead.

float **drag_bottom_margin** = `0.2` 🔗

- void **set_drag_margin**(margin: Side, drag_margin: float)
- float **get_drag_margin**(margin: Side) const

Bottom margin needed to drag the camera. A value of `1` makes the camera move only when reaching the bottom edge of the screen.

bool **drag_horizontal_enabled** = `false` 🔗

- void **set_drag_horizontal_enabled**(value: bool)
- bool **is_drag_horizontal_enabled**()

If `true`, the camera only moves when reaching the horizontal (left and right) drag margins. If `false`, the camera moves horizontally regardless of margins.

---

float **drag_horizontal_offset** = `0.0` 🔗

- void **set_drag_horizontal_offset**(value: float)
- float **get_drag_horizontal_offset**()

The relative horizontal drag offset of the camera between the right (`-1`) and left (`1`) drag margins.

Note: Used to set the initial horizontal drag offset; determine the current offset; or force the current offset. It's not automatically updated when drag_horizontal_enabled is `true` or the drag margins are changed.

---

float **drag_left_margin** = `0.2` 🔗

- void **set_drag_margin**(margin: Side, drag_margin: float)
- float **get_drag_margin**(margin: Side) const

Left margin needed to drag the camera. A value of `1` makes the camera move only when reaching the left edge of the screen.

---

float **drag_right_margin** = `0.2` 🔗

- void **set_drag_margin**(margin: Side, drag_margin: float)
- float **get_drag_margin**(margin: Side) const

Right margin needed to drag the camera. A value of `1` makes the camera move only when reaching the right edge of the screen.

---

float **drag_top_margin** = `0.2` 🔗

- void **set_drag_margin**(margin: Side, drag_margin: float)
- float **get_drag_margin**(margin: Side) const

Top margin needed to drag the camera. A value of `1` makes the camera move only when reaching the top edge of the screen.

---

bool **drag_vertical_enabled** = `false` 🔗

- void **set_drag_vertical_enabled**(value: bool)
- bool **is_drag_vertical_enabled**()

If `true`, the camera only moves when reaching the vertical (top and bottom) drag margins. If `false`, the camera moves vertically regardless of the drag margins.

---

float **drag_vertical_offset** = `0.0` 🔗

- void **set_drag_vertical_offset**(value: float)
- float **get_drag_vertical_offset**()

The relative vertical drag offset of the camera between the bottom (`-1`) and top (`1`) drag margins.

Note: Used to set the initial vertical drag offset; determine the current offset; or force the current offset. It's not automatically updated when drag_vertical_enabled is `true` or the drag margins are changed.

---

bool **editor_draw_drag_margin** = `false`

- void **set_margin_drawing_enabled**(value: bool)
- bool **is_margin_drawing_enabled**()

If `true`, draws the camera's drag margin rectangle in the editor.

---

bool **editor_draw_limits** = `false`

- void **set_limit_drawing_enabled**(value: bool)
- bool **is_limit_drawing_enabled**()

If `true`, draws the camera's limits rectangle in the editor.

---

bool **editor_draw_screen** = `true`

- void **set_screen_drawing_enabled**(value: bool)
- bool **is_screen_drawing_enabled**()

If `true`, draws the camera's screen rectangle in the editor.

---

bool **enabled** = `true`

- void **set_enabled**(value: bool)
- bool **is_enabled**()

Controls whether the camera can be active or not. If `true`, the **Camera2D** will become the main camera when it enters the scene tree and there is no active camera currently (see Viewport.get_camera_2d).

When the camera is currently active and enabled is set to `false`, the next enabled **Camera2D** in the scene tree will become active.

---

bool **ignore_rotation** = `true`

- void **set_ignore_rotation**(value: bool)
- bool **is_ignoring_rotation**()

If `true`, the camera's rendered view is not affected by its Node2D.rotation and Node2D.global_rotation.

---

int **limit_bottom** = `10000000`

- void **set_limit**(margin: Side, limit: int)
- int **get_limit**(margin: Side) const

Bottom scroll limit in pixels. The camera stops moving when reaching this value, but offset can push the view past the limit.

---

int **limit_left** = `-10000000`

- void **set_limit**(margin: Side, limit: int)
- int **get_limit**(margin: Side) const

Left scroll limit in pixels. The camera stops moving when reaching this value, but offset can push the view past the limit.

---

int **limit_right** = `10000000`

- void **set_limit**(margin: Side, limit: int)
- int **get_limit**(margin: Side) const

Right scroll limit in pixels. The camera stops moving when reaching this value, but offset can push the view past the limit.

---

bool **limit_smoothed** = `false`

- void **set_limit_smoothing_enabled**(value: bool)
- bool **is_limit_smoothing_enabled**()

If `true`, the camera smoothly stops when reaches its limits.

This property has no effect if position_smoothing_enabled is `false`.

Note: To immediately update the camera's position to be within limits without smoothing, even with this setting enabled, invoke reset_smoothing.

---

int **limit_top** = `-10000000`

- void **set_limit**(margin: Side, limit: int)
- int **get_limit**(margin: Side) const

Top scroll limit in pixels. The camera stops moving when reaching this value, but offset can push the view past the limit.

Vector2 **offset** = `Vector2(0, 0)`

- void **set_offset**(value: Vector2)
- Vector2 **get_offset**()

The camera's relative offset. Useful for looking around or camera shake animations. The offsetted camera can go past the limits defined in limit_top, limit_bottom, limit_left and limit_right.

---

bool **position_smoothing_enabled** = `false`

- void **set_position_smoothing_enabled**(value: bool)
- bool **is_position_smoothing_enabled**()

If `true`, the camera's view smoothly moves towards its target position at position_smoothing_speed.

float **position_smoothing_speed** = `5.0`

- void **set_position_smoothing_speed**(value: float)
- float **get_position_smoothing_speed**()

Speed in pixels per second of the camera's smoothing effect when position_smoothing_enabled is `true`.

Camera2DProcessCallback **process_callback** = `1`

- void **set_process_callback**(value: Camera2DProcessCallback)
- Camera2DProcessCallback **get_process_callback**()

The camera's process callback. See Camera2DProcessCallback.

bool **rotation_smoothing_enabled** = `false`

- void **set_rotation_smoothing_enabled**(value: bool)
- bool **is_rotation_smoothing_enabled**()

If `true`, the camera's view smoothly rotates, via asymptotic smoothing, to align with its target rotation at rotation_smoothing_speed.

Note: This property has no effect if ignore_rotation is `true`.

float **rotation_smoothing_speed** = `5.0`

- void **set_rotation_smoothing_speed**(value: float)
- float **get_rotation_smoothing_speed**()

The angular, asymptotic speed of the camera's rotation smoothing effect when rotation_smoothing_enabled is `true`.

Vector2 **zoom** = `Vector2(1, 1)` 🔗

- void **set_zoom**(value: Vector2)
- Vector2 **get_zoom**()

The camera's zoom. A zoom of `Vector(2, 2)` doubles the size seen in the viewport. A zoom of `Vector(0.5, 0.5)` halves the size seen in the viewport.

Note: FontFile.oversampling does *not* take **Camera2D** zoom into account. This means that zooming in/out will cause bitmap fonts and rasterized (non-MSDF) dynamic fonts to appear blurry or pixelated unless the font is part of a CanvasLayer that makes it ignore camera zoom. To ensure text remains crisp regardless of zoom, you can enable MSDF font rendering by enabling ProjectSettings.gui/theme/default_font_multichannel_signed_distance_field (applies to the default project font only), or enabling **Multichannel Signed Distance Field** in the import options of a DynamicFont for custom fonts. On system fonts, SystemFont.multichannel_signed_distance_field can be enabled in the inspector.

Method Descriptions

void **align**() 🔗

Aligns the camera to the tracked node.

---

void **force_update_scroll**() 🔗

Forces the camera to update scroll immediately.

float **get_drag_margin**(margin: Side) const

Returns the specified Side's margin. See also drag_bottom_margin, drag_top_margin, drag_left_margin, and drag_right_margin.

int **get_limit**(margin: Side) const

Returns the camera limit for the specified Side. See also limit_bottom, limit_top, limit_left, and limit_right.

Vector2 **get_screen_center_position**() const

Returns the center of the screen from this camera's point of view, in global coordinates.

Note: The exact targeted position of the camera may be different. See get_target_position.

Vector2 **get_target_position**() const

Returns this camera's target position, in global coordinates.

Note: The returned value is not the same as Node2D.global_position, as it is affected by the drag properties. It is also not the same as the current position if position_smoothing_enabled is `true` (see get_screen_center_position).

bool **is_current**() const

Returns `true` if this **Camera2D** is the active camera (see Viewport.get_camera_2d).

void **make_current**()

Forces this **Camera2D** to become the current active one. enabled must be `true`.

void **reset_smoothing**()

Sets the camera's position immediately to its current smoothing destination.

This method has no effect if position_smoothing_enabled is `false`.

void **set_drag_margin**(margin: Side, drag_margin: float)

Sets the specified Side's margin. See also drag_bottom_margin, drag_top_margin, drag_left_margin, and drag_right_margin.

void **set_limit**(margin: Side, limit: int)

Sets the camera limit for the specified Side. See also limit_bottom, limit_top, limit_left, and limit_right.

Camera3D

Inherits: Node3D **<** Node **<** Object

Inherited By: XRCamera3D

Camera node, displays from a point of view.

Description

Camera3D is a special node that displays what is visible from its current location. Cameras register themselves in the nearest Viewport node (when ascending the tree). Only one camera can be active per viewport. If no viewport is available ascending the tree, the camera will register in the global viewport. In other words, a camera just provides 3D display capabilities to a Viewport, and, without one, a scene registered in that Viewport (or higher viewports) can't be displayed.

Tutorials

- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

CameraAttributes	attributes
Compositor	compositor

int	cull_mask	`1048575`
bool	current	`false`
DopplerTracking	doppler_tracking	`0`
Environment	environment	
float	far	`4000.0`
float	fov	`75.0`
Vector2	frustum_offset	`Vector2(0, 0)`
float	h_offset	`0.0`
KeepAspect	keep_aspect	`1`
float	near	`0.05`
ProjectionType	projection	`0`
float	size	`1.0`

float	v_offset	`0.0`

Methods

void	clear_current(enable_next: bool = true)
Projection	get_camera_projection() const
RID	get_camera_rid() const
Transform3D	get_camera_transform() const
bool	get_cull_mask_value(layer_number: int) const
Array[Plane]	get_frustum() const
RID	get_pyramid_shape_rid()
bool	is_position_behind(world_point: Vector3) const
bool	is_position_in_frustum(world_point: Vector3) const
void	make_current()

Vector3	project_local_ray_normal(screen_point: Vector2) const
Vector3	project_position(screen_point: Vector2, z_depth: float) const
Vector3	project_ray_normal(screen_point: Vector2) const
Vector3	project_ray_origin(screen_point: Vector2) const
void	set_cull_mask_value(layer_number: int, value: bool)
void	set_frustum(size: float, offset: Vector2, z_near: float, z_far: float)
void	set_orthogonal(size: float, z_near: float, z_far: float)
void	set_perspective(fov: float, z_near: float, z_far: float)
Vector2	unproject_position(world_point: Vector3) const

Enumerations

enum **ProjectionType**: 🔗

ProjectionType **PROJECTION_PERSPECTIVE** = `0`

Perspective projection. Objects on the screen becomes smaller when they are far away.

ProjectionType **PROJECTION_ORTHOGONAL** = `1`

Orthogonal projection, also known as orthographic projection. Objects remain the same size on the screen no matter how far away they are.

ProjectionType **PROJECTION_FRUSTUM** = `2`

Frustum projection. This mode allows adjusting frustum_offset to create "tilted frustum" effects.

enum **KeepAspect**: 🔗

KeepAspect **KEEP_WIDTH** = `0`

Preserves the horizontal aspect ratio; also known as Vert- scaling. This is usually the best option for projects running in portrait mode, as taller aspect ratios will benefit from a wider vertical FOV.

KeepAspect **KEEP_HEIGHT** = `1`

Preserves the vertical aspect ratio; also known as Hor+ scaling. This is usually the best option for projects running in landscape mode, as wider aspect ratios will automatically benefit from a wider horizontal FOV.

enum **DopplerTracking**: 🔗

DopplerTracking **DOPPLER_TRACKING_DISABLED** = `0`

Disables Doppler effect [https://en.wikipedia.org/wiki/Doppler_effect] simulation (default).

DopplerTracking **DOPPLER_TRACKING_IDLE_STEP** = `1`

Simulate Doppler effect [https://en.wikipedia.org/wiki/Doppler_effect] by tracking positions of objects that are changed in `_process`. Changes in the relative velocity of this camera compared to those objects affect how audio is perceived (changing the audio's AudioStreamPlayer3D.pitch_scale).

DopplerTracking **DOPPLER_TRACKING_PHYSICS_STEP** = `2`

Simulate Doppler effect [https://en.wikipedia.org/wiki/Doppler_effect] by tracking positions of objects that are changed in `_physics_process`. Changes in the relative velocity of this camera compared to those objects affect how audio is perceived (changing the audio's AudioStreamPlayer3D.pitch_scale).

Property Descriptions

CameraAttributes **attributes**

- void **set_attributes**(value: CameraAttributes)
- CameraAttributes **get_attributes**()

The CameraAttributes to use for this camera.

Compositor **compositor**

- void **set_compositor**(value: Compositor)
- Compositor **get_compositor**()

The Compositor to use for this camera.

---

int **cull_mask** = `1048575`

- void **set_cull_mask**(value: int)
- int **get_cull_mask**()

The culling mask that describes which VisualInstance3D.layers are rendered by this camera. By default, all 20 user-visible layers are rendered.

Note: Since the cull_mask allows for 32 layers to be stored in total, there are an additional 12 layers that are only used internally by the engine and aren't exposed in the editor. Setting cull_mask using a script allows you to toggle those reserved layers, which can be useful for editor plugins.

To adjust cull_mask more easily using a script, use get_cull_mask_value and set_cull_mask_value.

Note: VoxelGI, SDFGI and LightmapGI will always take all layers into account to determine what contributes to global illumination. If this is an issue, set GeometryInstance3D.gi_mode to GeometryInstance3D.GI_MODE_DISABLED for meshes and Light3D.light_bake_mode to Light3D.BAKE_DISABLED for lights to exclude them from global illumination.

---

bool **current** = `false` 🔗

- void **set_current**(value: bool)
- bool **is_current**()

If `true`, the ancestor Viewport is currently using this camera.

If multiple cameras are in the scene, one will always be made current. For example, if two **Camera3D** nodes are present in the scene and only one is current, setting one camera's current to `false` will cause the other camera to be made current.

DopplerTracking **doppler_tracking** = `0`

- void **set_doppler_tracking**(value: DopplerTracking)
- DopplerTracking **get_doppler_tracking**()

If not DOPPLER_TRACKING_DISABLED, this camera will simulate the Doppler effect [https://en.wikipedia.org/wiki/Doppler_effect] for objects changed in particular `_process` methods. See DopplerTracking for possible values.

Environment **environment**

- void **set_environment**(value: Environment)
- Environment **get_environment**()

The Environment to use for this camera.

---

float **far** = `4000.0`

- void **set_far**(value: float)
- float **get_far**()

The distance to the far culling boundary for this camera relative to its local Z axis. Higher values allow the camera to see further away, while decreasing far can improve performance if it results in objects being partially or fully culled.

---

float **fov** = `75.0`

- void **set_fov**(value: float)
- float **get_fov**()

The camera's field of view angle (in degrees). Only applicable in perspective mode. Since keep_aspect locks one axis, fov sets the other axis' field of view angle.

For reference, the default vertical field of view value (`75.0`) is equivalent to a horizontal FOV of:

- ~91.31 degrees in a 4:3 viewport
- ~101.67 degrees in a 16:10 viewport
- ~107.51 degrees in a 16:9 viewport
- ~121.63 degrees in a 21:9 viewport

Vector2 **frustum_offset** = `Vector2(0, 0)` 🔗

- void **set_frustum_offset**(value: Vector2)
- Vector2 **get_frustum_offset**()

The camera's frustum offset. This can be changed from the default to create "tilted frustum" effects such as Y-shearing [https://zdoom.org/wiki/Y-shearing].

Note: Only effective if projection is PROJECTION_FRUSTUM.

---

float **h_offset** = `0.0` 🔗

- void **set_h_offset**(value: float)
- float **get_h_offset**()

The horizontal (X) offset of the camera viewport.

KeepAspect **keep_aspect** = `1` 🔗

- void **set_keep_aspect_mode**(value: KeepAspect)
- KeepAspect **get_keep_aspect_mode**()

The axis to lock during fov/size adjustments. Can be either KEEP_WIDTH or KEEP_HEIGHT.

---

float **near** = `0.05`

- void **set_near**(value: float)
- float **get_near**()

The distance to the near culling boundary for this camera relative to its local Z axis. Lower values allow the camera to see objects more up close to its origin, at the cost of lower precision across the *entire* range. Values lower than the default can lead to increased Z-fighting.

ProjectionType **projection** = `0`

- void **set_projection**(value: ProjectionType)
- ProjectionType **get_projection**()

The camera's projection mode. In PROJECTION_PERSPECTIVE mode, objects' Z distance from the camera's local space scales their perceived size.

---

float **size** = `1.0`

- void **set_size**(value: float)
- float **get_size**()

The camera's size in meters measured as the diameter of the width or height, depending on keep_aspect. Only applicable in orthogonal and frustum modes.

---

float **v_offset** = `0.0`

- void **set_v_offset**(value: float)
- float **get_v_offset**()

The vertical (Y) offset of the camera viewport.

Method Descriptions

void **clear_current**(enable_next: bool = true) 🔗

If this is the current camera, remove it from being current. If `enable_next` is `true`, request to make the next camera current, if any.

Projection **get_camera_projection**() const 🔗

Returns the projection matrix that this camera uses to render to its associated viewport. The camera must be part of the scene tree to function.

RID **get_camera_rid**() const 🔗

Returns the camera's RID from the RenderingServer.

Transform3D **get_camera_transform**() const 🔗

Returns the transform of the camera plus the vertical (v_offset) and horizontal (h_offset) offsets; and any other adjustments made to the position and orientation of the camera by subclassed cameras such as XRCamera3D.

bool **get_cull_mask_value**(layer_number: int) const 🔗

Returns whether or not the specified layer of the cull_mask is enabled, given a `layer_number` between 1 and 20.

Array[Plane] **get_frustum**() const 🔗

Returns the camera's frustum planes in world space units as an array of Planes in the following order: near, far, left, top, right, bottom. Not to be confused with frustum_offset.

RID **get_pyramid_shape_rid**() 🔗

Returns the RID of a pyramid shape encompassing the camera's view frustum, ignoring the camera's near plane. The tip of the pyramid represents the position of the camera.

---

bool **is_position_behind**(world_point: Vector3) const 🔗

Returns `true` if the given position is behind the camera (the blue part of the linked diagram). See this diagram [https://raw.githubusercontent.com/godotengine/godot-docs/master/img/camera3d_position_frustum.png] for an overview of position query methods.

Note: A position which returns `false` may still be outside the camera's field of view.

---

bool **is_position_in_frustum**(world_point: Vector3) const 🔗

Returns `true` if the given position is inside the camera's frustum (the green part of the linked diagram). See this diagram [https://raw.githubusercontent.com/godotengine/godot-docs/master/img/camera3d_position_frustum.png] for an overview of position query methods.

---

void **make_current**()

Makes this camera the current camera for the Viewport (see class description). If the camera node is outside the scene tree, it will attempt to become current once it's added.

Vector3 **project_local_ray_normal**(screen_point: Vector2) const

Returns a normal vector from the screen point location directed along the camera. Orthogonal cameras are normalized. Perspective cameras account for perspective, screen width/height, etc.

Vector3 **project_position**(screen_point: Vector2, z_depth: float) const

Returns the 3D point in world space that maps to the given 2D coordinate in the Viewport rectangle on a plane that is the given `z_depth` distance into the scene away from the camera.

Vector3 **project_ray_normal**(screen_point: Vector2) const

Returns a normal vector in world space, that is the result of projecting a point on the Viewport rectangle by the inverse camera projection. This is useful for casting rays in the form of (origin, normal) for object intersection or picking.

Vector3 **project_ray_origin**(screen_point: Vector2) const

Returns a 3D position in world space, that is the result of projecting a point on the Viewport rectangle by the inverse camera projection. This is useful for casting rays in the form of (origin, normal) for object intersection or picking.

---

void **set_cull_mask_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the cull_mask, given a `layer_number` between 1 and 20.

---

void **set_frustum**(size: float, offset: Vector2, z_near: float, z_far: float)

Sets the camera projection to frustum mode (see PROJECTION_FRUSTUM), by specifying a `size`, an `offset`, and the `z_near` and `z_far` clip planes in world space units. See also frustum_offset.

---

void **set_orthogonal**(size: float, z_near: float, z_far: float)

Sets the camera projection to orthogonal mode (see PROJECTION_ORTHOGONAL), by specifying a `size`, and the `z_near` and `z_far` clip planes in world space units. (As a hint, 2D games often use this projection, with values specified in pixels.)

---

void **set_perspective**(fov: float, z_near: float, z_far: float)

Sets the camera projection to perspective mode (see PROJECTION_PERSPECTIVE), by specifying a `fov` (field of view) angle in degrees, and the `z_near` and `z_far` clip planes in world space units.

Vector2 **unproject_position**(world_point: Vector3) const

Returns the 2D coordinate in the Viewport rectangle that maps to the given 3D point in world space.

Note: When using this to position GUI elements over a 3D viewport, use is_position_behind to prevent them from appearing if the 3D point is behind the camera:

```
# This code block is part of a script that inherits from Node3D.
# `control` is a reference to a node inheriting from Control.
control.visible = not get_viewport().get_camera_3d().is_position_behind(global_transform.origin)
control.position = get_viewport().get_camera_3d().unproject_position(global_transform.origin)
```

CanvasGroup

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Merges several 2D nodes into a single draw operation.

Description

Child CanvasItem nodes of a **CanvasGroup** are drawn as a single object. It allows to e.g. draw overlapping translucent 2D nodes without blending (set CanvasItem.self_modulate property of **CanvasGroup** to achieve this effect).

Note: The **CanvasGroup** uses a custom shader to read from the backbuffer to draw its children. Assigning a Material to the **CanvasGroup** overrides the builtin shader. To duplicate the behavior of the builtin shader in a custom Shader use the following:

```
shader_type canvas_item;
render_mode unshaded;

uniform sampler2D screen_texture : hint_screen_texture, repeat_disable, filter_nearest;

void fragment() {
    vec4 c = textureLod(screen_texture, SCREEN_UV, 0.0);

    if (c.a > 0.0001) {
        c.rgb /= c.a;
    }

    COLOR *= c;
}
```

Note: Since **CanvasGroup** and CanvasItem.clip_children both utilize the backbuffer, children of a **CanvasGroup** who have their CanvasItem.clip_children set to anything other than CanvasItem.CLIP_CHILDREN_DISABLED will not function correctly.

Properties

float	clear_margin	`10.0`
float	fit_margin	`10.0`
bool	use_mipmaps	`false`

Property Descriptions

float **clear_margin** = `10.0`

- void **set_clear_margin**(value: float)
- float **get_clear_margin**()

Sets the size of the margin used to expand the clearing rect of this **CanvasGroup**. This expands the area of the backbuffer that will be used by the **CanvasGroup**. A smaller margin will reduce the area of the backbuffer used which can increase performance, however if use_mipmaps is enabled, a small margin may result in mipmap errors at the edge of the **CanvasGroup**. Accordingly, this should be left as small as possible, but should be increased if artifacts appear along the edges of the canvas group.

---

float **fit_margin** = `10.0` 🔗

- void **set_fit_margin**(value: float)
- float **get_fit_margin**()

Sets the size of a margin used to expand the drawable rect of this **CanvasGroup**. The size of the **CanvasGroup** is determined by fitting a rect around its children then expanding that rect by fit_margin. This increases both the backbuffer area used and the area covered by the **CanvasGroup** both of which can reduce performance. This should be kept as small as possible and should only be expanded when an increased size is needed (e.g. for custom shader effects).

---

bool **use_mipmaps** = `false` 🔗

- void **set_use_mipmaps**(value: bool)
- bool **is_using_mipmaps**()

If `true`, calculates mipmaps for the backbuffer before drawing the **CanvasGroup** so that mipmaps can be used in a custom ShaderMaterial attached to the **CanvasGroup**. Generating mipmaps has a performance cost so this should not be enabled unless required.

CanvasItem

Inherits: Node **<** Object

Inherited By: Control, Node2D

Abstract base class for everything in 2D space.

Description

Abstract base class for everything in 2D space. Canvas items are laid out in a tree; children inherit and extend their parent's transform. **CanvasItem** is extended by Control for GUI-related nodes, and by Node2D for 2D game objects.

Any **CanvasItem** can draw. For this, queue_redraw is called by the engine, then NOTIFICATION_DRAW will be received on idle time to request a redraw. Because of this, canvas items don't need to be redrawn on every frame, improving the performance significantly. Several functions for drawing on the **CanvasItem** are provided (see `draw_*` functions). However, they can only be used inside _draw, its corresponding Object._notification or methods connected to the draw signal.

Canvas items are drawn in tree order on their canvas layer. By default, children are on top of their parents, so a root **CanvasItem** will be drawn behind everything. This behavior can be changed on a per-item basis.

A **CanvasItem** can be hidden, which will also hide its children. By adjusting various other properties of a **CanvasItem**, you can also modulate its color (via modulate or self_modulate), change its Z-index, blend mode, and more.

Note that properties like transform, modulation, and visibility are only propagated to *direct* **CanvasItem** child nodes. If there is a non-**CanvasItem** node in between, like Node or AnimationPlayer, the **CanvasItem** nodes below will have an independent position and modulate chain. See also top_level.

Tutorials

- Viewport and canvas transforms
- Custom drawing in 2D
- Audio Spectrum Visualizer Demo [https://godotengine.org/asset-library/asset/2762]

Properties

ClipChildrenMode	clip_children	`0`
int	light_mask	`1`
Material	material	
Color	modulate	`Color(1, 1, 1, 1)`
Color	self_modulate	`Color(1, 1, 1, 1)`
bool	show_behind_parent	`false`
TextureFilter	texture_filter	`0`

TextureRepeat	texture_repeat	`0`
bool	top_level	`false`
bool	use_parent_material	`false`
int	visibility_layer	`1`
bool	visible	`true`
bool	y_sort_enabled	`false`
bool	z_as_relative	`true`
int	z_index	`0`

Methods

void	_draw() virtual
void	draw_animation_slice(animation_length: float, slice_begin: float, slice_end: float, offset: float = 0.0)

void	draw_arc(center: Vector2, radius: float, start_angle: float, end_angle: float, point_count: int, color: Color, width: float = -1.0, antialiased: bool = false)
void	draw_char(font: Font, pos: Vector2, char: String, font_size: int = 16, modulate: Color = Color(1, 1, 1, 1)) const
void	draw_char_outline(font: Font, pos: Vector2, char: String, font_size: int = 16, size: int = -1, modulate: Color = Color(1, 1, 1, 1)) const
void	draw_circle(position: Vector2, radius: float, color: Color, filled: bool = true, width: float = -1.0, antialiased: bool = false)
void	draw_colored_polygon(points: PackedVector2Array, color: Color, uvs: PackedVector2Array = PackedVector2Array(), texture: Texture2D = null)
void	draw_dashed_line(from: Vector2, to: Vector2, color: Color, width: float = -1.0, dash: float = 2.0, aligned: bool = true, antialiased: bool = false)
void	draw_end_animation()
void	draw_lcd_texture_rect_region(texture: Texture2D, rect: Rect2, src_rect: Rect2, modulate: Color = Color(1, 1, 1, 1))

void	draw_line(from: Vector2, to: Vector2, color: Color, width: float = -1.0, antialiased: bool = false)
void	draw_mesh(mesh: Mesh, texture: Texture2D, transform: Transform2D = Transform2D(1, 0, 0, 1, 0, 0), modulate: Color = Color(1, 1, 1, 1))
void	draw_msdf_texture_rect_region(texture: Texture2D, rect: Rect2, src_rect: Rect2, modulate: Color = Color(1, 1, 1, 1), outline: float = 0.0, pixel_range: float = 4.0, scale: float = 1.0)
void	draw_multiline(points: PackedVector2Array, color: Color, width: float = -1.0, antialiased: bool = false)
void	draw_multiline_colors(points: PackedVector2Array, colors: PackedColorArray, width: float = -1.0, antialiased: bool = false)
void	draw_multiline_string(font: Font, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, max_lines: int = -1, modulate: Color = Color(1, 1, 1, 1), brk_flags: BitField[LineBreakFlag] = 3, justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const
void	draw_multiline_string_outline(font: Font, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16,

	max_lines: int = -1, size: int = 1, modulate: Color = Color(1, 1, 1, 1), brk_flags: BitField[LineBreakFlag] = 3, justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const
void	draw_multimesh(multimesh: MultiMesh, texture: Texture2D)
void	draw_polygon(points: PackedVector2Array, colors: PackedColorArray, uvs: PackedVector2Array = PackedVector2Array(), texture: Texture2D = null)
void	draw_polyline(points: PackedVector2Array, color: Color, width: float = -1.0, antialiased: bool = false)
void	draw_polyline_colors(points: PackedVector2Array, colors: PackedColorArray, width: float = -1.0, antialiased: bool = false)
void	draw_primitive(points: PackedVector2Array, colors: PackedColorArray, uvs: PackedVector2Array, texture: Texture2D = null)
void	draw_rect(rect: Rect2, color: Color, filled: bool = true, width: float = -1.0, antialiased: bool = false)
void	draw_set_transform(position: Vector2, rotation: float = 0.0, scale: Vector2 = Vector2(1, 1))

void	draw_set_transform_matrix(xform: Transform2D)
void	draw_string(font: Font, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, modulate: Color = Color(1, 1, 1, 1), justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const
void	draw_string_outline(font: Font, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, size: int = 1, modulate: Color = Color(1, 1, 1, 1), justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const
void	draw_style_box(style_box: StyleBox, rect: Rect2)
void	draw_texture(texture: Texture2D, position: Vector2, modulate: Color = Color(1, 1, 1, 1))
void	draw_texture_rect(texture: Texture2D, rect: Rect2, tile: bool, modulate: Color = Color(1, 1, 1, 1), transpose: bool = false)
void	draw_texture_rect_region(texture: Texture2D, rect: Rect2, src_rect: Rect2, modulate: Color = Color(1, 1, 1, 1), transpose: bool = false, clip_uv: bool = true)

void	force_update_transform()
RID	get_canvas() const
RID	get_canvas_item() const
CanvasLayer	get_canvas_layer_node() const
Transform2D	get_canvas_transform() const
Vector2	get_global_mouse_position() const
Transform2D	get_global_transform() const
Transform2D	get_global_transform_with_canvas() const
Vector2	get_local_mouse_position() const
Transform2D	get_screen_transform() const
Transform2D	get_transform() const
Rect2	get_viewport_rect() const
Transform2D	get_viewport_transform() const

bool	get_visibility_layer_bit(layer: int) const
World2D	get_world_2d() const
void	hide()
bool	is_local_transform_notification_enabled() const
bool	is_transform_notification_enabled() const
bool	is_visible_in_tree() const
Vector2	make_canvas_position_local(screen_point: Vector2) const
InputEvent	make_input_local(event: InputEvent) const
void	move_to_front()
void	queue_redraw()
void	set_notify_local_transform(enable: bool)
void	set_notify_transform(enable: bool)

void	set_visibility_layer_bit(layer: int, enabled: bool)
void	show()

Signals

draw()

Emitted when the **CanvasItem** must redraw, *after* the related NOTIFICATION_DRAW notification, and *before* _draw is called.

Note: Deferred connections do not allow drawing through the `draw_*` methods.

---

hidden()

Emitted when becoming hidden.

---

item_rect_changed()

Emitted when the item's Rect2 boundaries (position or size) have changed, or when an action is taking place that may have impacted these boundaries (e.g. changing Sprite2D.texture).

---

visibility_changed()

Emitted when the visibility (hidden/visible) changes.

Enumerations

enum **TextureFilter**: 🔗

TextureFilter **TEXTURE_FILTER_PARENT_NODE** = `0`

The **CanvasItem** will inherit the filter from its parent.

TextureFilter **TEXTURE_FILTER_NEAREST** = `1`

The texture filter reads from the nearest pixel only. This makes the texture look pixelated from up close, and grainy from a distance (due to mipmaps not being sampled).

TextureFilter **TEXTURE_FILTER_LINEAR** = `2`

The texture filter blends between the nearest 4 pixels. This makes the texture look smooth from up close, and grainy from a distance (due to mipmaps not being sampled).

TextureFilter **TEXTURE_FILTER_NEAREST_WITH_MIPMAPS** = `3`

The texture filter reads from the nearest pixel and blends between the nearest 2 mipmaps (or uses the nearest mipmap if ProjectSettings.rendering/textures/default_filters/use_nearest_mipmap_filter is `true`). This makes the texture look pixelated from up close, and smooth from a distance.

Use this for non-pixel art textures that may be viewed at a low scale (e.g. due to Camera2D zoom or sprite scaling), as mipmaps are important to smooth out pixels that are smaller than on-screen pixels.

TextureFilter **TEXTURE_FILTER_LINEAR_WITH_MIPMAPS** = `4`

The texture filter blends between the nearest 4 pixels and between the nearest 2 mipmaps (or uses the nearest mipmap if ProjectSettings.rendering/textures/default_filters/use_nearest_mipmap_filter is `true`). This makes the texture look smooth from up close, and smooth from a distance.

Use this for non-pixel art textures that may be viewed at a low scale (e.g. due to Camera2D zoom or sprite scaling), as mipmaps are important to smooth out pixels that are smaller than on-screen pixels.

TextureFilter **TEXTURE_FILTER_NEAREST_WITH_MIPMAPS_ANISOTROPIC** = `5`

The texture filter reads from the nearest pixel and blends between 2 mipmaps (or uses the nearest mipmap if ProjectSettings.rendering/textures/default_filters/use_nearest_mipmap_filter is `true`) based on the angle between the surface and the camera view. This makes the texture look pixelated from up close, and smooth from a distance. Anisotropic filtering improves texture quality on surfaces that are almost in line with the camera, but is slightly slower. The anisotropic filtering level can be changed by adjusting ProjectSettings.rendering/textures/default_filters/anisotropic_filtering_level.

Note: This texture filter is rarely useful in 2D projects. TEXTURE_FILTER_NEAREST_WITH_MIPMAPS is usually more appropriate in this case.

TextureFilter **TEXTURE_FILTER_LINEAR_WITH_MIPMAPS_ANISOTROPIC** = `6`

The texture filter blends between the nearest 4 pixels and blends between 2 mipmaps (or uses the nearest mipmap if ProjectSettings.rendering/textures/default_filters/use_nearest_mipmap_filter is `true`) based on the angle between the surface and the camera view. This makes the texture look smooth from up close, and smooth from a distance. Anisotropic filtering improves texture quality on surfaces that are almost in line with the camera, but is slightly slower. The anisotropic filtering level can be changed by adjusting ProjectSettings.rendering/textures/default_filters/anisotropic_filtering_level.

Note: This texture filter is rarely useful in 2D projects. TEXTURE_FILTER_LINEAR_WITH_MIPMAPS is usually more appropriate in this case.

TextureFilter **TEXTURE_FILTER_MAX** = `7`

Represents the size of the TextureFilter enum.

---

enum **TextureRepeat**: 🔗

TextureRepeat **TEXTURE_REPEAT_PARENT_NODE** = `0`

The **CanvasItem** will inherit the filter from its parent.

TextureRepeat **TEXTURE_REPEAT_DISABLED** = `1`

Texture will not repeat.

TextureRepeat **TEXTURE_REPEAT_ENABLED** = `2`

Texture will repeat normally.

TextureRepeat **TEXTURE_REPEAT_MIRROR** = `3`

Texture will repeat in a 2×2 tiled mode, where elements at even positions are mirrored.

TextureRepeat **TEXTURE_REPEAT_MAX** = `4`

Represents the size of the TextureRepeat enum.

---

enum **ClipChildrenMode**:

ClipChildrenMode **CLIP_CHILDREN_DISABLED** = `0`

Child draws over parent and is not clipped.

ClipChildrenMode **CLIP_CHILDREN_ONLY** = `1`

Parent is used for the purposes of clipping only. Child is clipped to the parent's visible area, parent is not drawn.

ClipChildrenMode **CLIP_CHILDREN_AND_DRAW** = `2`

Parent is used for clipping child, but parent is also drawn underneath child as normal before clipping child to its visible area.

ClipChildrenMode **CLIP_CHILDREN_MAX** = `3`

Represents the size of the ClipChildrenMode enum.

Constants

NOTIFICATION_TRANSFORM_CHANGED = `2000`

The **CanvasItem**'s global transform has changed. This notification is only received if enabled by set_notify_transform.

NOTIFICATION_LOCAL_TRANSFORM_CHANGED = `35`

The **CanvasItem**'s local transform has changed. This notification is only received if enabled by set_notify_local_transform.

NOTIFICATION_DRAW = `30`

The **CanvasItem** is requested to draw (see _draw).

NOTIFICATION_VISIBILITY_CHANGED = `31`

The **CanvasItem**'s visibility has changed.

NOTIFICATION_ENTER_CANVAS = `32`

The **CanvasItem** has entered the canvas.

NOTIFICATION_EXIT_CANVAS = `33`

The **CanvasItem** has exited the canvas.

NOTIFICATION_WORLD_2D_CHANGED = `36`

The **CanvasItem**'s active World2D changed.

Property Descriptions

ClipChildrenMode **clip_children** = `0`

- void **set_clip_children_mode**(value: ClipChildrenMode)
- ClipChildrenMode **get_clip_children_mode**()

Allows the current node to clip child nodes, essentially acting as a mask.

---

int **light_mask** = `1`

- void **set_light_mask**(value: int)
- int **get_light_mask**()

The rendering layers in which this **CanvasItem** responds to Light2D nodes.

Material **material** 🔗

- void **set_material**(value: Material)
- Material **get_material**()

The material applied to this **CanvasItem**.

Color **modulate** = `Color(1, 1, 1, 1)` 🔗

- void **set_modulate**(value: Color)
- Color **get_modulate**()

The color applied to this **CanvasItem**. This property does affect child **CanvasItem**s, unlike self_modulate which only affects the node itself.

Color **self_modulate** = `Color(1, 1, 1, 1)` 🔗

- void **set_self_modulate**(value: Color)
- Color **get_self_modulate**()

The color applied to this **CanvasItem**. This property does **not** affect child **CanvasItem**s, unlike modulate which affects both the node itself and its children.

Note: Internal children (e.g. sliders in ColorPicker or tab bar in TabContainer) are also not affected by this property (see `include_internal` parameter of Node.get_child and other similar methods).

---

bool **show_behind_parent** = `false`

- void **set_draw_behind_parent**(value: bool)
- bool **is_draw_behind_parent_enabled**()

If `true`, the object draws behind its parent.

TextureFilter **texture_filter** = `0`

- void **set_texture_filter**(value: TextureFilter)
- TextureFilter **get_texture_filter**()

The texture filtering mode to use on this **CanvasItem**.

TextureRepeat **texture_repeat** = `0`

- void **set_texture_repeat**(value: TextureRepeat)
- TextureRepeat **get_texture_repeat**()

The texture repeating mode to use on this **CanvasItem**.

---

bool **top_level** = `false`

- void **set_as_top_level**(value: bool)
- bool **is_set_as_top_level**()

If `true`, this **CanvasItem** will *not* inherit its transform from parent **CanvasItem**s. Its draw order will also be changed to make it draw on top of other **CanvasItem**s that do not have top_level set to `true`. The **CanvasItem** will effectively act as if it was placed as a child of a bare Node.

---

bool **use_parent_material** = `false`

- void **set_use_parent_material**(value: bool)
- bool **get_use_parent_material**()

If `true`, the parent **CanvasItem**'s material property is used as this one's material.

---

int **visibility_layer** = `1`

- void **set_visibility_layer**(value: int)
- int **get_visibility_layer**()

The rendering layer in which this **CanvasItem** is rendered by Viewport nodes. A Viewport will render a **CanvasItem** if it and all its parents share a layer with the Viewport's canvas cull mask.

---

bool **visible** = `true`

- void **set_visible**(value: bool)
- bool **is_visible**()

If `true`, this **CanvasItem** is drawn. The node is only visible if all of its ancestors are visible as well (in other words, is_visible_in_tree must return `true`).

Note: For controls that inherit Popup, the correct way to make them visible is to call one of the multiple `popup*()` functions instead.

---

bool **y_sort_enabled** = `false`

- void **set_y_sort_enabled**(value: bool)
- bool **is_y_sort_enabled**()

If `true`, this and child **CanvasItem** nodes with a higher Y position are rendered in front of nodes with a lower Y position. If `false`, this and child **CanvasItem** nodes are rendered normally in scene tree order.

With Y-sorting enabled on a parent node ('A') but disabled on a child node ('B'), the child node ('B') is sorted but its children ('C1', 'C2', etc) render together on the same Y position as the child node ('B'). This allows you to organize the render order of a scene without changing the scene tree.

Nodes sort relative to each other only if they are on the same z_index.

---

bool **z_as_relative** = `true` 🔗

- void **set_z_as_relative**(value: bool)
- bool **is_z_relative**()

If `true`, the node's Z index is relative to its parent's Z index. If this node's Z index is 2 and its parent's effective Z index is 3, then this node's effective Z index will be 2 + 3 = 5.

---

int **z_index** = `0` 🔗

- void **set_z_index**(value: int)
- int **get_z_index**()

Controls the order in which the nodes render. A node with a higher Z index will display in front of others. Must be between RenderingServer.CANVAS_ITEM_Z_MIN and RenderingServer.CANVAS_ITEM_Z_MAX (inclusive).

Note: Changing the Z index of a Control only affects the drawing order, not the order in which input events are handled. This can be useful to implement certain UI animations, e.g. a menu where hovered items are scaled and should overlap others.

Method Descriptions

void **_draw**() virtual 🔗

Called when **CanvasItem** has been requested to redraw (after queue_redraw is called, either manually or by the engine).

Corresponds to the NOTIFICATION_DRAW notification in Object._notification.

---

void **draw_animation_slice**(animation_length: float, slice_begin: float, slice_end: float, offset: float = 0.0) 🔗

Subsequent drawing commands will be ignored unless they fall within the specified animation slice. This is a faster way to implement animations that loop on background rather than redrawing constantly.

---

void **draw_arc**(center: Vector2, radius: float, start_angle: float, end_angle: float, point_count: int, color: Color, width: float = -1.0, antialiased: bool = false) 🔗

Draws an unfilled arc between the given angles with a uniform `color` and `width` and optional antialiasing (supported only for positive `width`). The larger the value of `point_count`, the smoother the curve. See also draw_circle.

If `width` is negative, it will be ignored and the arc will be drawn using RenderingServer.PRIMITIVE_LINE_STRIP. This means that when the CanvasItem is scaled, the arc will remain thin. If this behavior is not desired, then pass a positive `width` like `1.0`.

The arc is drawn from `start_angle` towards the value of `end_angle` so in clockwise direction if `start_angle < end_angle` and counter-clockwise otherwise. Passing the same angles but in reversed order will produce the same arc. If absolute difference of `start_angle` and `end_angle` is greater than @GDScript.TAU radians, then a full circle arc is drawn (i.e. arc will not overlap itself).

---

void **draw_char**(font: Font, pos: Vector2, char: String, font_size: int = 16, modulate: Color = Color(1, 1, 1, 1)) const 🔗

Draws a string first character using a custom font.

---

void **draw_char_outline**(font: Font, pos: Vector2, char: String, font_size: int = 16, size: int = -1, modulate: Color = Color(1, 1, 1, 1)) const 🔗

Draws a string first character outline using a custom font.

---

void **draw_circle**(position: Vector2, radius: float, color: Color, filled: bool = true, width: float = -1.0, antialiased: bool = false) 🔗

Draws a circle. See also draw_arc, draw_polyline, and draw_polygon.

If `filled` is `true`, the circle will be filled with the `color` specified. If `filled` is `false`, the circle will be drawn as a stroke with the `color` and `width` specified.

If `width` is negative, then two-point primitives will be drawn instead of a four-point ones. This means that when the CanvasItem is scaled, the lines will remain thin. If this behavior is not desired, then pass a positive `width` like `1.0`.

If `antialiased` is `true`, half transparent "feathers" will be attached to the boundary, making outlines smooth.

Note: `width` is only effective if `filled` is `false`.

---

void **draw_colored_polygon**(points: PackedVector2Array, color: Color, uvs: PackedVector2Array = PackedVector2Array(), texture: Texture2D = null) 🔗

Draws a colored polygon of any number of points, convex or concave. Unlike draw_polygon, a single color must be specified for the whole polygon.

---

void **draw_dashed_line**(from: Vector2, to: Vector2, color: Color, width: float = -1.0, dash: float = 2.0, aligned: bool = true, antialiased: bool = false) 🔗

Draws a dashed line from a 2D point to another, with a given color and width. See also draw_multiline and draw_polyline.

If `width` is negative, then a two-point primitives will be drawn instead of a four-point ones. This means that when the CanvasItem is scaled, the line parts will remain thin. If this behavior is not desired, then pass a positive `width` like `1.0`.

If `antialiased` is `true`, half transparent "feathers" will be attached to the boundary, making outlines smooth.

Note: `antialiased` is only effective if `width` is greater than `0.0`.

---

void **draw_end_animation**()

After submitting all animations slices via draw_animation_slice, this function can be used to revert drawing to its default state (all subsequent drawing commands will be visible). If you don't care about this particular use case, usage of this function after submitting the slices is not required.

---

void **draw_lcd_texture_rect_region**(texture: Texture2D, rect: Rect2, src_rect: Rect2, modulate: Color = Color(1, 1, 1, 1))

Draws a textured rectangle region of the font texture with LCD subpixel anti-aliasing at a given position, optionally modulated by a color.

Texture is drawn using the following blend operation, blend mode of the CanvasItemMaterial is ignored:

```
dst.r = texture.r * modulate.r * modulate.a + dst.r * (1.0 - texture.r * modulate.a);
dst.g = texture.g * modulate.g * modulate.a + dst.g * (1.0 - texture.g * modulate.a);
dst.b = texture.b * modulate.b * modulate.a + dst.b * (1.0 - texture.b * modulate.a);
dst.a = modulate.a + dst.a * (1.0 - modulate.a);
```

---

void **draw_line**(from: Vector2, to: Vector2, color: Color, width: float = -1.0, antialiased: bool = false)

Draws a line from a 2D point to another, with a given color and width. It can be optionally antialiased. See also draw_multiline and draw_polyline.

If `width` is negative, then a two-point primitive will be drawn instead of a four-point one. This means that when the CanvasItem is scaled, the line will remain thin. If this behavior is not desired, then pass a positive `width` like `1.0`.

void **draw_mesh**(mesh: Mesh, texture: Texture2D, transform: Transform2D = Transform2D(1, 0, 0, 1, 0, 0), modulate: Color = Color(1, 1, 1, 1)) 🔗

Draws a Mesh in 2D, using the provided texture. See MeshInstance2D for related documentation.

void **draw_msdf_texture_rect_region**(texture: Texture2D, rect: Rect2, src_rect: Rect2, modulate: Color = Color(1, 1, 1, 1), outline: float = 0.0, pixel_range: float = 4.0, scale: float = 1.0) 🔗

Draws a textured rectangle region of the multi-channel signed distance field texture at a given position, optionally modulated by a color. See FontFile.multichannel_signed_distance_field for more information and caveats about MSDF font rendering.

If `outline` is positive, each alpha channel value of pixel in region is set to maximum value of true distance in the `outline` radius.

Value of the `pixel_range` should the same that was used during distance field texture generation.

void **draw_multiline**(points: PackedVector2Array, color: Color, width: float = -1.0, antialiased: bool = false) 🔗

Draws multiple disconnected lines with a uniform `width` and `color`. Each line is defined by two consecutive points from `points` array, i.e. i-th segment consists of `points[2 * i]`, `points[2 * i + 1]` endpoints. When drawing large amounts of lines, this is faster than using individual draw_line calls. To draw interconnected lines, use draw_polyline instead.

If `width` is negative, then two-point primitives will be drawn instead of a four-point ones. This means that when the CanvasItem is scaled, the lines will remain thin. If this behavior is not desired, then pass a positive `width` like `1.0`.

Note: `antialiased` is only effective if `width` is greater than `0.0`.

---

void **draw_multiline_colors**(points: PackedVector2Array, colors: PackedColorArray, width: float = -1.0, antialiased: bool = false) 🔗

Draws multiple disconnected lines with a uniform `width` and segment-by-segment coloring. Each segment is defined by two consecutive points from `points` array and a corresponding color from `colors` array, i.e. i-th segment consists of `points[2 * i]`, `points[2 * i + 1]` endpoints and has `colors[i]` color. When drawing large amounts of lines, this is faster than using individual draw_line calls. To draw interconnected lines, use draw_polyline_colors instead.

If `width` is negative, then two-point primitives will be drawn instead of a four-point ones. This means that when the CanvasItem is scaled, the lines will remain thin. If this behavior is not desired, then pass a positive `width` like `1.0`.

Note: `antialiased` is only effective if `width` is greater than `0.0`.

---

void **draw_multiline_string**(font: Font, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, max_lines: int = -1, modulate: Color = Color(1, 1, 1, 1), brk_flags: BitField[LineBreakFlag] = 3, justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const 🔗

Breaks `text` into lines and draws it using the specified `font` at the `pos` (top-left corner). The text will have its color multiplied by `modulate`. If `width` is greater than or equal to 0, the text will be clipped if it exceeds the specified width.

---

void **draw_multiline_string_outline**(font: Font, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, max_lines: int = -1, size: int = 1, modulate: Color = Color(1, 1, 1, 1), brk_flags: BitField[LineBreakFlag] = 3, justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const 🔗

Breaks `text` to the lines and draws text outline using the specified `font` at the `pos` (top-left corner). The text will have its color multiplied by `modulate`. If `width` is greater than or equal to 0, the text will be clipped if it exceeds the specified width.

---

void **draw_multimesh**(multimesh: MultiMesh, texture: Texture2D) 🔗

Draws a MultiMesh in 2D with the provided texture. See MultiMeshInstance2D for related documentation.

---

void **draw_polygon**(points: PackedVector2Array, colors: PackedColorArray, uvs: PackedVector2Array = PackedVector2Array(), texture: Texture2D = null) 🔗

Draws a solid polygon of any number of points, convex or concave. Unlike draw_colored_polygon, each point's color can be changed individually. See also draw_polyline and draw_polyline_colors. If you need more flexibility (such as being able to use bones), use RenderingServer.canvas_item_add_triangle_array instead.

---

void **draw_polyline**(points: PackedVector2Array, color: Color, width: float = -1.0, antialiased: bool = false) 🔗

Draws interconnected line segments with a uniform `color` and `width` and optional antialiasing (supported only for positive `width`). When drawing large amounts of lines, this is faster than using individual draw_line calls. To draw disconnected lines, use draw_multiline instead. See also draw_polygon.

If `width` is negative, it will be ignored and the polyline will be drawn using RenderingServer.PRIMITIVE_LINE_STRIP. This means that when the CanvasItem is scaled, the polyline will remain thin. If this behavior is not desired, then pass a positive `width` like `1.0`.

---

void **draw_polyline_colors**(points: PackedVector2Array, colors: PackedColorArray, width: float = -1.0, antialiased: bool = false) 🔗

Draws interconnected line segments with a uniform `width`, point-by-point coloring, and optional antialiasing (supported only for positive `width`). Colors assigned to line points match by index between `points` and `colors`, i.e. each line segment is filled with a gradient between the colors of the endpoints. When drawing large amounts of lines, this is faster than using individual draw_line calls. To draw disconnected lines, use draw_multiline_colors instead. See also draw_polygon.

If `width` is negative, it will be ignored and the polyline will be drawn using RenderingServer.PRIMITIVE_LINE_STRIP. This means that when the CanvasItem is scaled, the polyline will remain thin. If this behavior is not desired, then pass a positive `width` like `1.0`.

---

void **draw_primitive**(points: PackedVector2Array, colors: PackedColorArray, uvs: PackedVector2Array, texture: Texture2D = null) 🔗

Draws a custom primitive. 1 point for a point, 2 points for a line, 3 points for a triangle, and 4 points for a quad. If 0 points or more than 4 points are specified, nothing will be drawn and an error message will be printed. See also draw_line, draw_polyline, draw_polygon, and draw_rect.

void **draw_rect**(rect: Rect2, color: Color, filled: bool = true, width: float = -1.0, antialiased: bool = false)

Draws a rectangle. If `filled` is `true`, the rectangle will be filled with the `color` specified. If `filled` is `false`, the rectangle will be drawn as a stroke with the `color` and `width` specified. See also draw_texture_rect.

If `width` is negative, then two-point primitives will be drawn instead of a four-point ones. This means that when the CanvasItem is scaled, the lines will remain thin. If this behavior is not desired, then pass a positive `width` like `1.0`.

If `antialiased` is `true`, half transparent "feathers" will be attached to the boundary, making outlines smooth.

Note: `width` is only effective if `filled` is `false`.

Note: Unfilled rectangles drawn with a negative `width` may not display perfectly. For example, corners may be missing or brighter due to overlapping lines (for a translucent `color`).

void **draw_set_transform**(position: Vector2, rotation: float = 0.0, scale: Vector2 = Vector2(1, 1))

Sets a custom transform for drawing via components. Anything drawn afterwards will be transformed by this.

Note: FontFile.oversampling does *not* take `scale` into account. This means that scaling up/down will cause bitmap fonts and rasterized (non-MSDF) dynamic fonts to appear blurry or pixelated. To ensure text remains crisp regardless of scale, you can enable MSDF font rendering by enabling ProjectSettings.gui/theme/default_font_multichannel_signed_distance_field (applies to the default project font only), or enabling **Multichannel Signed Distance Field** in the import options of a DynamicFont for custom fonts. On system fonts, SystemFont.multichannel_signed_distance_field can be enabled in the inspector.

---

void **draw_set_transform_matrix**(xform: Transform2D) 🔗

Sets a custom transform for drawing via matrix. Anything drawn afterwards will be transformed by this.

---

void **draw_string**(font: Font, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, modulate: Color = Color(1, 1, 1, 1), justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const 🔗

Draws `text` using the specified `font` at the `pos` (bottom-left corner using the baseline of the font). The text will have its color multiplied by `modulate`. If `width` is greater than or equal to 0, the text will be clipped if it exceeds the specified width.

Example using the default project font:

GDScript

```
# If using this method in a script that redraws constantly,
move the
# `default_font` declaration to a member variable assigned in
```

```
`_ready()`
# so the Control is only created once.
var default_font = ThemeDB.fallback_font
var default_font_size = ThemeDB.fallback_font_size
draw_string(default_font, Vector2(64, 64), "Hello world",
HORIZONTAL_ALIGNMENT_LEFT, -1, default_font_size)
```

C#

```
// If using this method in a script that redraws constantly,
move the
// `default_font` declaration to a member variable assigned in
`_Ready()`
// so the Control is only created once.
Font defaultFont = ThemeDB.FallbackFont;
int defaultFontSize = ThemeDB.FallbackFontSize;
DrawString(defaultFont, new Vector2(64, 64), "Hello world",
HORIZONTAL_ALIGNMENT_LEFT, -1, defaultFontSize);
```

See also Font.draw_string.

---

void **draw_string_outline**(font: Font, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, size: int = 1, modulate: Color = Color(1, 1, 1, 1), justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const 🔗

Draws `text` outline using the specified `font` at the `pos` (bottom-left corner using the baseline of the font). The text will have its color multiplied by `modulate`. If `width` is greater than or equal to 0, the text will be clipped if it exceeds the specified width.

---

void **draw_style_box**(style_box: StyleBox, rect: Rect2) 🔗

Draws a styled rectangle.

---

void **draw_texture**(texture: Texture2D, position: Vector2, modulate: Color = Color(1, 1, 1, 1)) 🔗

Draws a texture at a given position.

void **draw_texture_rect**(texture: Texture2D, rect: Rect2, tile: bool, modulate: Color = Color(1, 1, 1, 1), transpose: bool = false) 🔗

Draws a textured rectangle at a given position, optionally modulated by a color. If `transpose` is `true`, the texture will have its X and Y coordinates swapped. See also draw_rect and draw_texture_rect_region.

void **draw_texture_rect_region**(texture: Texture2D, rect: Rect2, src_rect: Rect2, modulate: Color = Color(1, 1, 1, 1), transpose: bool = false, clip_uv: bool = true) 🔗

Draws a textured rectangle from a texture's region (specified by `src_rect`) at a given position, optionally modulated by a color. If `transpose` is `true`, the texture will have its X and Y coordinates swapped. See also draw_texture_rect.

void **force_update_transform**() 🔗

Forces the transform to update. Transform changes in physics are not instant for performance reasons. Transforms are accumulated and then set. Use this if you need an up-to-date transform when doing physics operations.

RID **get_canvas**() const 🔗

Returns the RID of the World2D canvas where this item is in.

RID **get_canvas_item**() const

Returns the canvas item RID used by RenderingServer for this item.

CanvasLayer **get_canvas_layer_node**() const

Returns the CanvasLayer that contains this node, or `null` if the node is not in any CanvasLayer.

Transform2D **get_canvas_transform**() const

Returns the transform from the coordinate system of the canvas, this item is in, to the Viewports coordinate system.

Vector2 **get_global_mouse_position**() const

Returns the mouse's position in the CanvasLayer that this **CanvasItem** is in using the coordinate system of the CanvasLayer.

Note: For screen-space coordinates (e.g. when using a non-embedded Popup), you can use DisplayServer.mouse_get_position.

Transform2D **get_global_transform**() const

Returns the global transform matrix of this item, i.e. the combined transform up to the topmost **CanvasItem** node. The topmost item is a **CanvasItem** that either has no parent, has non-**CanvasItem** parent or it has top_level enabled.

Transform2D **get_global_transform_with_canvas**() const

Returns the transform from the local coordinate system of this **CanvasItem** to the Viewports coordinate system.

Vector2 **get_local_mouse_position**() const 🔗

Returns the mouse's position in this **CanvasItem** using the local coordinate system of this **CanvasItem**.

Transform2D **get_screen_transform**() const 🔗

Returns the transform of this **CanvasItem** in global screen coordinates (i.e. taking window position into account). Mostly useful for editor plugins.

Equals to get_global_transform if the window is embedded (see Viewport.gui_embed_subwindows).

Transform2D **get_transform**() const 🔗

Returns the transform matrix of this item.

Rect2 **get_viewport_rect**() const 🔗

Returns the viewport's boundaries as a Rect2.

Transform2D **get_viewport_transform**() const 🔗

Returns the transform from the coordinate system of the canvas, this item is in, to the Viewports embedders coordinate system.

---

bool **get_visibility_layer_bit**(layer: int) const 🔗

Returns an individual bit on the rendering visibility layer.

World2D **get_world_2d**() const 🔗

Returns the World2D where this item is in.

---

void **hide**() 🔗

Hide the **CanvasItem** if it's currently visible. This is equivalent to setting visible to `false`.

---

bool **is_local_transform_notification_enabled**() const 🔗

Returns `true` if local transform notifications are communicated to children.

---

bool **is_transform_notification_enabled**() const 🔗

Returns `true` if global transform notifications are communicated to children.

---

bool **is_visible_in_tree**() const 🔗

Returns `true` if the node is present in the SceneTree, its visible property is `true` and all its ancestors are also visible. If any ancestor is hidden, this node will not be visible in the scene tree, and is therefore not drawn (see _draw).

Visibility is checked only in parent nodes that inherit from **CanvasItem**, CanvasLayer, and Window. If the parent is of any other type (such as Node, AnimationPlayer, or Node3D), it is assumed to be visible.

Vector2 **make_canvas_position_local**(screen_point: Vector2) const

Assigns `screen_point` as this node's new local transform.

InputEvent **make_input_local**(event: InputEvent) const

Transformations issued by `event`'s inputs are applied in local space instead of global space.

void **move_to_front**()

Moves this node to display on top of its siblings.

Internally, the node is moved to the bottom of parent's child list. The method has no effect on nodes without a parent.

void **queue_redraw**()

Queues the **CanvasItem** to redraw. During idle time, if **CanvasItem** is visible, NOTIFICATION_DRAW is sent and _draw is called. This only occurs **once** per frame, even if this method has been called multiple times.

void **set_notify_local_transform**(enable: bool)

If `enable` is `true`, this node will receive NOTIFICATION_LOCAL_TRANSFORM_CHANGED when its local transform changes.

void **set_notify_transform**(enable: bool) 🔗

If `enable` is `true`, this node will receive NOTIFICATION_TRANSFORM_CHANGED when its global transform changes.

---

void **set_visibility_layer_bit**(layer: int, enabled: bool) 🔗

Set/clear individual bits on the rendering visibility layer. This simplifies editing this **CanvasItem**'s visibility layer.

---

void **show**() 🔗

Show the **CanvasItem** if it's currently hidden. This is equivalent to setting visible to `true`. For controls that inherit Popup, the correct way to make them visible is to call one of the multiple `popup*()` functions instead.

CanvasLayer

Inherits: Node **<** Object

Inherited By: ParallaxBackground

A node used for independent rendering of objects within a 2D scene.

Description

CanvasItem-derived nodes that are direct or indirect children of a **CanvasLayer** will be drawn in that layer. The layer is a numeric index that defines the draw order. The default 2D scene renders with index `0`, so a **CanvasLayer** with index `-1` will be drawn below, and a **CanvasLayer** with index `1` will be drawn above. This order will hold regardless of the CanvasItem.z_index of the nodes within each layer.

CanvasLayers can be hidden and they can also optionally follow the viewport. This makes them useful for HUDs like health bar overlays (on layers `1` and higher) or backgrounds (on layers `-1` and lower).

Note: Embedded Windows are placed on layer `1024`. CanvasItems on layers `1025` and higher appear in front of embedded windows.

Note: Each **CanvasLayer** is drawn on one specific Viewport and cannot be shared between multiple Viewports, see custom_viewport. When using multiple Viewports, for example in a split-screen game, you need create an individual **CanvasLayer** for each Viewport you want it to be drawn on.

Tutorials

- Viewport and canvas transforms
- Canvas layers
- 2D Dodge The Creeps Demo [https://godotengine.org/asset-library/asset/2712]

Properties

Node	custom_viewport	
bool	follow_viewport_enabled	`false`
float	follow_viewport_scale	`1.0`
int	layer	`1`
Vector2	offset	`Vector2(0, 0)`
float	rotation	`0.0`
Vector2	scale	`Vector2(1, 1)`
Transform2D	transform	`Transform2D(1, 0, 0, 1, 0, 0)`
bool	visible	`true`

Methods

RID	get_canvas() const
Transform2D	get_final_transform() const
void	hide()
void	show()

Signals

visibility_changed()

Emitted when visibility of the layer is changed. See visible.

Property Descriptions

Node **custom_viewport**

- void **set_custom_viewport**(value: Node)
- Node **get_custom_viewport**()

The custom Viewport node assigned to the **CanvasLayer**. If `null`, uses the default viewport instead.

bool **follow_viewport_enabled** = `false` 🔗

- void **set_follow_viewport**(value: bool)
- bool **is_following_viewport**()

If enabled, the **CanvasLayer** will use the viewport's transform, so it will move when camera moves instead of being anchored in a fixed position on the screen.

Together with follow_viewport_scale it can be used for a pseudo 3D effect.

---

float **follow_viewport_scale** = `1.0` 🔗

- void **set_follow_viewport_scale**(value: float)
- float **get_follow_viewport_scale**()

Scales the layer when using follow_viewport_enabled. Layers moving into the foreground should have increasing scales, while layers moving into the background should have decreasing scales.

---

int **layer** = `1` 🔗

- void **set_layer**(value: int)
- int **get_layer**()

Layer index for draw order. Lower values are drawn behind higher values.

Note: If multiple CanvasLayers have the same layer index, CanvasItem children of one CanvasLayer are drawn behind the CanvasItem children of the other CanvasLayer. Which CanvasLayer is drawn in front is non-deterministic.

Vector2 **offset** = `Vector2(0, 0)`

- void **set_offset**(value: Vector2)
- Vector2 **get_offset**()

The layer's base offset.

---

float **rotation** = `0.0`

- void **set_rotation**(value: float)
- float **get_rotation**()

The layer's rotation in radians.

Vector2 **scale** = `Vector2(1, 1)`

- void **set_scale**(value: Vector2)
- Vector2 **get_scale**()

The layer's scale.

Transform2D **transform** = `Transform2D(1, 0, 0, 1, 0, 0)`

- void **set_transform**(value: Transform2D)
- Transform2D **get_transform**()

The layer's transform.

---

bool **visible** = `true`

- void **set_visible**(value: bool)
- bool **is_visible**()

If `false`, any CanvasItem under this **CanvasLayer** will be hidden.

Unlike CanvasItem.visible, visibility of a **CanvasLayer** isn't propagated to underlying layers.

Method Descriptions

RID **get_canvas**() const 🔗

Returns the RID of the canvas used by this layer.

Transform2D **get_final_transform**() const 🔗

Returns the transform from the **CanvasLayers** coordinate system to the Viewports coordinate system.

---

void **hide**() 🔗

Hides any CanvasItem under this **CanvasLayer**. This is equivalent to setting visible to `false`.

---

void **show**() 🔗

Shows any CanvasItem under this **CanvasLayer**. This is equivalent to setting visible to `true`.

CanvasModulate

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A node that applies a color tint to a canvas.

Description

CanvasModulate applies a color tint to all nodes on a canvas. Only one can be used to tint a canvas, but CanvasLayers can be used to render things independently.

Properties

Color	color	`Color(1, 1, 1, 1)`

Property Descriptions

Color **color** = `Color(1, 1, 1, 1)`

- void **set_color**(value: Color)
- Color **get_color**()

The tint color to apply.

CenterContainer

Inherits: Container **<** Control **<** CanvasItem **<** Node **<** Object

A container that keeps child controls in its center.

Description

CenterContainer is a container that keeps all of its child controls in its center at their minimum size.

Tutorials

- Using Containers

Properties

bool	use_top_left	`false`

Property Descriptions

bool **use_top_left** = `false` 🔗

- void **set_use_top_left**(value: bool)
- bool **is_using_top_left**()

If `true`, centers children relative to the **CenterContainer**'s top left corner.

CharacterBody2D

Inherits: PhysicsBody2D **<** CollisionObject2D **<** Node2D **<** CanvasItem **<** Node **<** Object

A 2D physics body specialized for characters moved by script.

Description

CharacterBody2D is a specialized class for physics bodies that are meant to be user-controlled. They are not affected by physics at all, but they affect other physics bodies in their path. They are mainly used to provide high-level API to move objects with wall and slope detection (move_and_slide method) in addition to the general collision detection provided by PhysicsBody2D.move_and_collide. This makes it useful for highly configurable physics bodies that must move in specific ways and collide with the world, as is often the case with user-controlled characters.

For game objects that don't require complex movement or collision detection, such as moving platforms, AnimatableBody2D is simpler to configure.

Tutorials

- Kinematic character (2D)
- Using CharacterBody2D
- 2D Kinematic Character Demo [https://godotengine.org/asset-library/asset/2719]
- 2D Platformer Demo [https://godotengine.org/asset-library/asset/2727]

Properties

bool	floor_block_on_wall	`true`
bool	floor_constant_speed	`false`
float	floor_max_angle	`0.785398`
float	floor_snap_length	`1.0`
bool	floor_stop_on_slope	`true`
int	max_slides	`4`
MotionMode	motion_mode	`0`
int	platform_floor_layers	`4294967295`
PlatformOnLeave	platform_on_leave	`0`
int	platform_wall_layers	`0`
float	safe_margin	`0.08`
bool	slide_on_ceiling	`true`

Vector2	up_direction	`Vector2(0, -1)`
Vector2	velocity	`Vector2(0, 0)`
float	wall_min_slide_angle	`0.261799`

Methods

void	apply_floor_snap()
float	get_floor_angle(up_direction: Vector2 = Vector2(0, -1)) const
Vector2	get_floor_normal() const
Vector2	get_last_motion() const
KinematicCollision2D	get_last_slide_collision()
Vector2	get_platform_velocity() const
Vector2	get_position_delta() const
Vector2	get_real_velocity() const

KinematicCollision2D	get_slide_collision(slide_idx: int)
int	get_slide_collision_count() const
Vector2	get_wall_normal() const
bool	is_on_ceiling() const
bool	is_on_ceiling_only() const
bool	is_on_floor() const
bool	is_on_floor_only() const
bool	is_on_wall() const
bool	is_on_wall_only() const
bool	move_and_slide()

Enumerations

enum **MotionMode**: 🔗

MotionMode **MOTION_MODE_GROUNDED** = `0`

Apply when notions of walls, ceiling and floor are relevant. In this mode the body motion will react to slopes (acceleration/slowdown). This mode is suitable for sided games like platformers.

MotionMode **MOTION_MODE_FLOATING** = `1`

Apply when there is no notion of floor or ceiling. All collisions will be reported as `on_wall`. In this mode, when you slide, the speed will always be constant. This mode is suitable for top-down games.

---

enum **PlatformOnLeave**: 🔗

PlatformOnLeave **PLATFORM_ON_LEAVE_ADD_VELOCITY** = `0`

Add the last platform velocity to the velocity when you leave a moving platform.

PlatformOnLeave **PLATFORM_ON_LEAVE_ADD_UPWARD_VELOCITY** = `1`

Add the last platform velocity to the velocity when you leave a moving platform, but any downward motion is ignored. It's useful to keep full jump height even when the platform is moving down.

PlatformOnLeave **PLATFORM_ON_LEAVE_DO_NOTHING** = `2`

Do nothing when leaving a platform.

Property Descriptions

bool **floor_block_on_wall** = `true` 🔗

- void **set_floor_block_on_wall_enabled**(value: bool)
- bool **is_floor_block_on_wall_enabled**()

If `true`, the body will be able to move on the floor only. This option avoids to be able to walk on walls, it will however allow to slide down along them.

---

bool **floor_constant_speed** = `false` 🔗

- void **set_floor_constant_speed_enabled**(value: bool)
- bool **is_floor_constant_speed_enabled**()

If `false` (by default), the body will move faster on downward slopes and slower on upward slopes.

If `true`, the body will always move at the same speed on the ground no matter the slope. Note that you need to use floor_snap_length to stick along a downward slope at constant speed.

---

float **floor_max_angle** = `0.785398` 🔗

- void **set_floor_max_angle**(value: float)
- float **get_floor_max_angle**()

Maximum angle (in radians) where a slope is still considered a floor (or a ceiling), rather than a wall, when calling move_and_slide. The default value equals 45 degrees.

---

float **floor_snap_length** = `1.0` 🔗

- void **set_floor_snap_length**(value: float)
- float **get_floor_snap_length**()

Sets a snapping distance. When set to a value different from `0.0`, the body is kept attached to slopes when calling move_and_slide. The snapping vector is determined by the given distance along the opposite direction of the up_direction.

As long as the snapping vector is in contact with the ground and the body moves against up_direction, the body will remain attached to the surface. Snapping is not applied if the body moves along up_direction, meaning it contains vertical rising velocity, so it will be able to detach from the ground when jumping or when the body is pushed up by something. If you want to apply a snap without taking into account the velocity, use apply_floor_snap.

---

bool **floor_stop_on_slope** = `true`

- void **set_floor_stop_on_slope_enabled**(value: bool)
- bool **is_floor_stop_on_slope_enabled**()

If `true`, the body will not slide on slopes when calling move_and_slide when the body is standing still.

If `false`, the body will slide on floor's slopes when velocity applies a downward force.

---

int **max_slides** = `4`

- void **set_max_slides**(value: int)
- int **get_max_slides**()

Maximum number of times the body can change direction before it stops when calling move_and_slide.

MotionMode **motion_mode** = `0`

- void **set_motion_mode**(value: MotionMode)
- MotionMode **get_motion_mode**()

Sets the motion mode which defines the behavior of move_and_slide. See MotionMode constants for available modes.

---

int **platform_floor_layers** = `4294967295` 🔗

- void **set_platform_floor_layers**(value: int)
- int **get_platform_floor_layers**()

Collision layers that will be included for detecting floor bodies that will act as moving platforms to be followed by the **CharacterBody2D**. By default, all floor bodies are detected and propagate their velocity.

PlatformOnLeave **platform_on_leave** = `0` 🔗

- void **set_platform_on_leave**(value: PlatformOnLeave)
- PlatformOnLeave **get_platform_on_leave**()

Sets the behavior to apply when you leave a moving platform. By default, to be physically accurate, when you leave the last platform velocity is applied. See PlatformOnLeave constants for available behavior.

---

int **platform_wall_layers** = `0` 🔗

- void **set_platform_wall_layers**(value: int)
- int **get_platform_wall_layers**()

Collision layers that will be included for detecting wall bodies that will act as moving platforms to be followed by the **CharacterBody2D**. By default, all wall bodies are ignored.

---

float **safe_margin** = `0.08` 🔗

- void **set_safe_margin**(value: float)
- float **get_safe_margin**()

Extra margin used for collision recovery when calling move_and_slide.

If the body is at least this close to another body, it will consider them to be colliding and will be pushed away before performing the actual motion.

A higher value means it's more flexible for detecting collision, which helps with consistently detecting walls and floors.

A lower value forces the collision algorithm to use more exact detection, so it can be used in cases that specifically require precision, e.g at very low scale to avoid visible jittering, or for stability with a stack of character bodies.

---

bool **slide_on_ceiling** = `true` 🔗

- void **set_slide_on_ceiling_enabled**(value: bool)
- bool **is_slide_on_ceiling_enabled**()

If `true`, during a jump against the ceiling, the body will slide, if `false` it will be stopped and will fall vertically.

Vector2 **up_direction** = `Vector2(0, -1)` 🔗

- void **set_up_direction**(value: Vector2)
- Vector2 **get_up_direction**()

Vector pointing upwards, used to determine what is a wall and what is a floor (or a ceiling) when calling move_and_slide. Defaults to Vector2.UP. As the vector will be normalized it can't be equal to Vector2.ZERO, if you want all collisions to be reported as walls, consider using MOTION_MODE_FLOATING as motion_mode.

Vector2 **velocity** = `Vector2(0, 0)` 🔗

- void **set_velocity**(value: Vector2)
- Vector2 **get_velocity**()

Current velocity vector in pixels per second, used and modified during calls to move_and_slide.

---

float **wall_min_slide_angle** = `0.261799` 🔗

- void **set_wall_min_slide_angle**(value: float)
- float **get_wall_min_slide_angle**()

Minimum angle (in radians) where the body is allowed to slide when it encounters a slope. The default value equals 15 degrees. This property only affects movement when motion_mode is MOTION_MODE_FLOATING.

Method Descriptions

void **apply_floor_snap**() 🔗

Allows to manually apply a snap to the floor regardless of the body's velocity. This function does nothing when is_on_floor returns `true`.

---

float **get_floor_angle**(up_direction: Vector2 = Vector2(0, -1)) const

Returns the floor's collision angle at the last collision point according to `up_direction`, which is Vector2.UP by default. This value is always positive and only valid after calling move_and_slide and when is_on_floor returns `true`.

Vector2 **get_floor_normal**() const

Returns the collision normal of the floor at the last collision point. Only valid after calling move_and_slide and when is_on_floor returns `true`.

Warning: The collision normal is not always the same as the surface normal.

Vector2 **get_last_motion**() const

Returns the last motion applied to the **CharacterBody2D** during the last call to move_and_slide. The movement can be split into multiple motions when sliding occurs, and this method return the last one, which is useful to retrieve the current direction of the movement.

KinematicCollision2D **get_last_slide_collision**()

Returns a KinematicCollision2D, which contains information about the latest collision that occurred during the last call to move_and_slide.

Vector2 **get_platform_velocity**() const

Returns the linear velocity of the platform at the last collision point. Only valid after calling move_and_slide.

Vector2 **get_position_delta**() const

Returns the travel (position delta) that occurred during the last call to move_and_slide.

Vector2 **get_real_velocity**() const

Returns the current real velocity since the last call to move_and_slide. For example, when you climb a slope, you will move diagonally even though the velocity is horizontal. This method returns the diagonal movement, as opposed to velocity which returns the requested velocity.

KinematicCollision2D **get_slide_collision**(slide_idx: int)

Returns a KinematicCollision2D, which contains information about a collision that occurred during the last call to move_and_slide. Since the body can collide several times in a single call to move_and_slide, you must specify the index of the collision in the range 0 to (get_slide_collision_count - 1).

Example usage:

GDScript

```
for i in get_slide_collision_count():
    var collision = get_slide_collision(i)
    print("Collided with: ", collision.get_collider().name)
```

C#

```
for (int i = 0; i < GetSlideCollisionCount(); i++)
{
    KinematicCollision2D collision = GetSlideCollision(i);
    GD.Print("Collided with: ", (collision.GetCollider() as
Node).Name);
}
```

int **get_slide_collision_count**() const

Returns the number of times the body collided and changed direction during the last call to move_and_slide.

Vector2 **get_wall_normal**() const

Returns the collision normal of the wall at the last collision point. Only valid after calling move_and_slide and when is_on_wall returns `true`.

Warning: The collision normal is not always the same as the surface normal.

bool **is_on_ceiling**() const

Returns `true` if the body collided with the ceiling on the last call of move_and_slide. Otherwise, returns `false`. The up_direction and floor_max_angle are used to determine whether a surface is "ceiling" or not.

bool **is_on_ceiling_only**() const

Returns `true` if the body collided only with the ceiling on the last call of move_and_slide. Otherwise, returns `false`. The up_direction and floor_max_angle are used to determine whether a surface is "ceiling" or not.

---

bool **is_on_floor**() const

Returns `true` if the body collided with the floor on the last call of move_and_slide. Otherwise, returns `false`. The up_direction and floor_max_angle are used to determine whether a surface is "floor" or not.

---

bool **is_on_floor_only**() const

Returns `true` if the body collided only with the floor on the last call of move_and_slide. Otherwise, returns `false`. The up_direction and floor_max_angle are used to determine whether a surface is "floor" or not.

---

bool **is_on_wall**() const

Returns `true` if the body collided with a wall on the last call of move_and_slide. Otherwise, returns `false`. The up_direction and floor_max_angle are used to determine whether a surface is "wall" or not.

---

bool **is_on_wall_only**() const

Returns `true` if the body collided only with a wall on the last call of move_and_slide. Otherwise, returns `false`. The up_direction and floor_max_angle are used to determine whether a surface is "wall" or not.

---

bool **move_and_slide**() 🔗

Moves the body based on velocity. If the body collides with another, it will slide along the other body (by default only on floor) rather than stop immediately. If the other body is a **CharacterBody2D** or RigidBody2D, it will also be affected by the motion of the other body. You can use this to make moving and rotating platforms, or to make nodes push other nodes.

Modifies velocity if a slide collision occurred. To get the latest collision call get_last_slide_collision, for detailed information about collisions that occurred, use get_slide_collision.

When the body touches a moving platform, the platform's velocity is automatically added to the body motion. If a collision occurs due to the platform's motion, it will always be first in the slide collisions.

The general behavior and available properties change according to the motion_mode.

Returns `true` if the body collided, otherwise, returns `false`.

CharacterBody3D

Inherits: PhysicsBody3D **<** CollisionObject3D **<** Node3D **<** Node **<** Object

A 3D physics body specialized for characters moved by script.

Description

CharacterBody3D is a specialized class for physics bodies that are meant to be user-controlled. They are not affected by physics at all, but they affect other physics bodies in their path. They are mainly used to provide high-level API to move objects with wall and slope detection (move_and_slide method) in addition to the general collision detection provided by PhysicsBody3D.move_and_collide. This makes it useful for highly configurable physics bodies that must move in specific ways and collide with the world, as is often the case with user-controlled characters.

For game objects that don't require complex movement or collision detection, such as moving platforms, AnimatableBody3D is simpler to configure.

Tutorials

- Kinematic character (2D)
- 3D Kinematic Character Demo [https://godotengine.org/asset-library/asset/2739]
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]
- 3D Voxel Demo [https://godotengine.org/asset-library/asset/2755]
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

bool	floor_block_on_wall	`true`
bool	floor_constant_speed	`false`
float	floor_max_angle	`0.785398`
float	floor_snap_length	`0.1`
bool	floor_stop_on_slope	`true`
int	max_slides	`6`
MotionMode	motion_mode	`0`
int	platform_floor_layers	`4294967295`
PlatformOnLeave	platform_on_leave	`0`
int	platform_wall_layers	`0`
float	safe_margin	`0.001`
bool	slide_on_ceiling	`true`

Vector3	up_direction	`Vector3(0, 1, 0)`
Vector3	velocity	`Vector3(0, 0, 0)`
float	wall_min_slide_angle	`0.261799`

Methods

void	apply_floor_snap()
float	get_floor_angle(up_direction: Vector3 = Vector3(0, 1, 0)) const
Vector3	get_floor_normal() const
Vector3	get_last_motion() const
KinematicCollision3D	get_last_slide_collision()
Vector3	get_platform_angular_velocity() const
Vector3	get_platform_velocity() const
Vector3	get_position_delta() const

Vector3	get_real_velocity() const
KinematicCollision3D	get_slide_collision(slide_idx: int)
int	get_slide_collision_count() const
Vector3	get_wall_normal() const
bool	is_on_ceiling() const
bool	is_on_ceiling_only() const
bool	is_on_floor() const
bool	is_on_floor_only() const
bool	is_on_wall() const
bool	is_on_wall_only() const
bool	move_and_slide()

Enumerations

enum **MotionMode**: 🔗

MotionMode **MOTION_MODE_GROUNDED** = `0`

Apply when notions of walls, ceiling and floor are relevant. In this mode the body motion will react to slopes (acceleration/slowdown). This mode is suitable for grounded games like platformers.

MotionMode **MOTION_MODE_FLOATING** = `1`

Apply when there is no notion of floor or ceiling. All collisions will be reported as `on_wall`. In this mode, when you slide, the speed will always be constant. This mode is suitable for games without ground like space games.

---

enum **PlatformOnLeave**: 🔗

PlatformOnLeave **PLATFORM_ON_LEAVE_ADD_VELOCITY** = `0`

Add the last platform velocity to the velocity when you leave a moving platform.

PlatformOnLeave **PLATFORM_ON_LEAVE_ADD_UPWARD_VELOCITY** = `1`

Add the last platform velocity to the velocity when you leave a moving platform, but any downward motion is ignored. It's useful to keep full jump height even when the platform is moving down.

PlatformOnLeave **PLATFORM_ON_LEAVE_DO_NOTHING** = `2`

Do nothing when leaving a platform.

Property Descriptions

bool **floor_block_on_wall** = `true` 🔗

- void **set_floor_block_on_wall_enabled**(value: bool)
- bool **is_floor_block_on_wall_enabled**()

If `true`, the body will be able to move on the floor only. This option avoids to be able to walk on walls, it will however allow to slide down along them.

---

bool **floor_constant_speed** = `false` 🔗

- void **set_floor_constant_speed_enabled**(value: bool)
- bool **is_floor_constant_speed_enabled**()

If `false` (by default), the body will move faster on downward slopes and slower on upward slopes.

If `true`, the body will always move at the same speed on the ground no matter the slope. Note that you need to use floor_snap_length to stick along a downward slope at constant speed.

---

float **floor_max_angle** = `0.785398` 🔗

- void **set_floor_max_angle**(value: float)
- float **get_floor_max_angle**()

Maximum angle (in radians) where a slope is still considered a floor (or a ceiling), rather than a wall, when calling move_and_slide. The default value equals 45 degrees.

---

float **floor_snap_length** = `0.1` 🔗

- void **set_floor_snap_length**(value: float)
- float **get_floor_snap_length**()

Sets a snapping distance. When set to a value different from `0.0`, the body is kept attached to slopes when calling move_and_slide. The snapping vector is determined by the given distance along the opposite direction of the up_direction.

As long as the snapping vector is in contact with the ground and the body moves against up_direction, the body will remain attached to the surface. Snapping is not applied if the body moves along up_direction, meaning it contains vertical rising velocity, so it will be able to detach from the ground when jumping or when the body is pushed up by something. If you want to apply a snap without taking into account the velocity, use apply_floor_snap.

bool **floor_stop_on_slope** = `true` 🔗

- void **set_floor_stop_on_slope_enabled**(value: bool)
- bool **is_floor_stop_on_slope_enabled**()

If `true`, the body will not slide on slopes when calling move_and_slide when the body is standing still.

If `false`, the body will slide on floor's slopes when velocity applies a downward force.

int **max_slides** = `6` 🔗

- void **set_max_slides**(value: int)
- int **get_max_slides**()

Maximum number of times the body can change direction before it stops when calling move_and_slide.

MotionMode **motion_mode** = `0` 🔗

- void **set_motion_mode**(value: MotionMode)
- MotionMode **get_motion_mode**()

Sets the motion mode which defines the behavior of move_and_slide. See MotionMode constants for available modes.

---

int **platform_floor_layers** = `4294967295`

- void **set_platform_floor_layers**(value: int)
- int **get_platform_floor_layers**()

Collision layers that will be included for detecting floor bodies that will act as moving platforms to be followed by the **CharacterBody3D**. By default, all floor bodies are detected and propagate their velocity.

PlatformOnLeave **platform_on_leave** = `0`

- void **set_platform_on_leave**(value: PlatformOnLeave)
- PlatformOnLeave **get_platform_on_leave**()

Sets the behavior to apply when you leave a moving platform. By default, to be physically accurate, when you leave the last platform velocity is applied. See PlatformOnLeave constants for available behavior.

---

int **platform_wall_layers** = `0`

- void **set_platform_wall_layers**(value: int)
- int **get_platform_wall_layers**()

Collision layers that will be included for detecting wall bodies that will act as moving platforms to be followed by the **CharacterBody3D**. By default, all wall bodies are ignored.

---

float **safe_margin** = `0.001`

- void **set_safe_margin**(value: float)
- float **get_safe_margin**()

Extra margin used for collision recovery when calling move_and_slide.

If the body is at least this close to another body, it will consider them to be colliding and will be pushed away before performing the actual motion.

A higher value means it's more flexible for detecting collision, which helps with consistently detecting walls and floors.

A lower value forces the collision algorithm to use more exact detection, so it can be used in cases that specifically require precision, e.g at very low scale to avoid visible jittering, or for stability with a stack of character bodies.

---

bool **slide_on_ceiling** = `true`

- void **set_slide_on_ceiling_enabled**(value: bool)
- bool **is_slide_on_ceiling_enabled**()

If `true`, during a jump against the ceiling, the body will slide, if `false` it will be stopped and will fall vertically.

Vector3 **up_direction** = `Vector3(0, 1, 0)`

- void **set_up_direction**(value: Vector3)
- Vector3 **get_up_direction**()

Vector pointing upwards, used to determine what is a wall and what is a floor (or a ceiling) when calling move_and_slide. Defaults to Vector3.UP. As the vector will be normalized it can't be equal to Vector3.ZERO, if you want all collisions to be reported as walls, consider using MOTION_MODE_FLOATING as motion_mode.

Vector3 **velocity** = `Vector3(0, 0, 0)` 🔗

- void **set_velocity**(value: Vector3)
- Vector3 **get_velocity**()

Current velocity vector (typically meters per second), used and modified during calls to move_and_slide.

---

float **wall_min_slide_angle** = `0.261799` 🔗

- void **set_wall_min_slide_angle**(value: float)
- float **get_wall_min_slide_angle**()

Minimum angle (in radians) where the body is allowed to slide when it encounters a slope. The default value equals 15 degrees. When motion_mode is MOTION_MODE_GROUNDED, it only affects movement if floor_block_on_wall is `true`.

Method Descriptions

void **apply_floor_snap**() 🔗

Allows to manually apply a snap to the floor regardless of the body's velocity. This function does nothing when is_on_floor returns `true`.

---

float **get_floor_angle**(up_direction: Vector3 = Vector3(0, 1, 0)) const

Returns the floor's collision angle at the last collision point according to `up_direction`, which is Vector3.UP by default. This value is always positive and only valid after calling move_and_slide and when is_on_floor returns `true`.

Vector3 **get_floor_normal**() const

Returns the collision normal of the floor at the last collision point. Only valid after calling move_and_slide and when is_on_floor returns `true`.

Warning: The collision normal is not always the same as the surface normal.

Vector3 **get_last_motion**() const

Returns the last motion applied to the **CharacterBody3D** during the last call to move_and_slide. The movement can be split into multiple motions when sliding occurs, and this method return the last one, which is useful to retrieve the current direction of the movement.

KinematicCollision3D **get_last_slide_collision**()

Returns a KinematicCollision3D, which contains information about the latest collision that occurred during the last call to move_and_slide.

Vector3 **get_platform_angular_velocity**() const

Returns the angular velocity of the platform at the last collision point. Only valid after calling move_and_slide.

Vector3 **get_platform_velocity**() const 🔗

Returns the linear velocity of the platform at the last collision point. Only valid after calling move_and_slide.

Vector3 **get_position_delta**() const 🔗

Returns the travel (position delta) that occurred during the last call to move_and_slide.

Vector3 **get_real_velocity**() const 🔗

Returns the current real velocity since the last call to move_and_slide. For example, when you climb a slope, you will move diagonally even though the velocity is horizontal. This method returns the diagonal movement, as opposed to velocity which returns the requested velocity.

KinematicCollision3D **get_slide_collision**(slide_idx: int) 🔗

Returns a KinematicCollision3D, which contains information about a collision that occurred during the last call to move_and_slide. Since the body can collide several times in a single call to move_and_slide, you must specify the index of the collision in the range 0 to (get_slide_collision_count - 1).

---

int **get_slide_collision_count**() const 🔗

Returns the number of times the body collided and changed direction during the last call to move_and_slide.

Vector3 **get_wall_normal**() const 🔗

Returns the collision normal of the wall at the last collision point. Only valid after calling move_and_slide and when is_on_wall returns `true`.

Warning: The collision normal is not always the same as the surface normal.

---

bool **is_on_ceiling**() const 🔗

Returns `true` if the body collided with the ceiling on the last call of move_and_slide. Otherwise, returns `false`. The up_direction and floor_max_angle are used to determine whether a surface is "ceiling" or not.

---

bool **is_on_ceiling_only**() const 🔗

Returns `true` if the body collided only with the ceiling on the last call of move_and_slide. Otherwise, returns `false`. The up_direction and floor_max_angle are used to determine whether a surface is "ceiling" or not.

---

bool **is_on_floor**() const 🔗

Returns `true` if the body collided with the floor on the last call of move_and_slide. Otherwise, returns `false`. The up_direction and floor_max_angle are used to determine whether a surface is "floor" or not.

---

bool **is_on_floor_only**() const

Returns `true` if the body collided only with the floor on the last call of move_and_slide. Otherwise, returns `false`. The up_direction and floor_max_angle are used to determine whether a surface is "floor" or not.

---

bool **is_on_wall**() const

Returns `true` if the body collided with a wall on the last call of move_and_slide. Otherwise, returns `false`. The up_direction and floor_max_angle are used to determine whether a surface is "wall" or not.

---

bool **is_on_wall_only**() const

Returns `true` if the body collided only with a wall on the last call of move_and_slide. Otherwise, returns `false`. The up_direction and floor_max_angle are used to determine whether a surface is "wall" or not.

---

bool **move_and_slide**()

Moves the body based on velocity. If the body collides with another, it will slide along the other body rather than stop immediately. If the other body is a **CharacterBody3D** or RigidBody3D, it will also be affected by the motion of the other body. You can use this to make moving and rotating platforms, or to make nodes push other nodes.

Modifies velocity if a slide collision occurred. To get the latest collision call get_last_slide_collision, for more detailed information about collisions that occurred, use get_slide_collision.

When the body touches a moving platform, the platform's velocity is automatically added to the body motion. If a collision occurs due to the platform's motion, it will always be first in the slide collisions.

Returns `true` if the body collided, otherwise, returns `false`.

CheckBox

Inherits: Button **<** BaseButton **<** Control **<** CanvasItem **<** Node **<** Object

A button that represents a binary choice.

Description

CheckBox allows the user to choose one of only two possible options. It's similar to CheckButton in functionality, but it has a different appearance. To follow established UX patterns, it's recommended to use **CheckBox** when toggling it has **no** immediate effect on something. For example, it could be used when toggling it will only do something once a confirmation button is pressed.

See also BaseButton which contains common properties and methods associated with this node.

When BaseButton.button_group specifies a ButtonGroup, **CheckBox** changes its appearance to that of a radio button and uses the various `radio_*` theme properties.

Properties

HorizontalAlignment	alignment	`0` (overrides Button)
bool	toggle_mode	`true` (overrides BaseButton)

Theme Properties

int	check_v_offset	0
Texture2D	checked	
Texture2D	checked_disabled	
Texture2D	radio_checked	
Texture2D	radio_checked_disabled	
Texture2D	radio_unchecked	
Texture2D	radio_unchecked_disabled	
Texture2D	unchecked	
Texture2D	unchecked_disabled	

Theme Property Descriptions

int **check_v_offset** = 0 🔗

The vertical offset used when rendering the check icons (in pixels).

Texture2D **checked**

The check icon to display when the **CheckBox** is checked.

Texture2D **checked_disabled**

The check icon to display when the **CheckBox** is checked and is disabled.

Texture2D **radio_checked**

The check icon to display when the **CheckBox** is configured as a radio button and is checked.

Texture2D **radio_checked_disabled**

The check icon to display when the **CheckBox** is configured as a radio button, is disabled, and is unchecked.

Texture2D **radio_unchecked**

The check icon to display when the **CheckBox** is configured as a radio button and is unchecked.

Texture2D **radio_unchecked_disabled**

The check icon to display when the **CheckBox** is configured as a radio button, is disabled, and is unchecked.

Texture2D **unchecked**

The check icon to display when the **CheckBox** is unchecked.

Texture2D **unchecked_disabled** 🔗

The check icon to display when the **CheckBox** is unchecked and is disabled.

CheckButton

Inherits: Button **<** BaseButton **<** Control **<** CanvasItem **<** Node **<** Object

A button that represents a binary choice.

Description

CheckButton is a toggle button displayed as a check field. It's similar to CheckBox in functionality, but it has a different appearance. To follow established UX patterns, it's recommended to use **CheckButton** when toggling it has an **immediate** effect on something. For example, it can be used when pressing it shows or hides advanced settings, without asking the user to confirm this action.

See also BaseButton which contains common properties and methods associated with this node.

Properties

HorizontalAlignment	alignment	`0` (overrides Button)
bool	toggle_mode	`true` (overrides BaseButton)

Theme Properties

int	check_v_offset	0
Texture2D	checked	
Texture2D	checked_disabled	
Texture2D	checked_disabled_mirrored	
Texture2D	checked_mirrored	
Texture2D	unchecked	
Texture2D	unchecked_disabled	
Texture2D	unchecked_disabled_mirrored	
Texture2D	unchecked_mirrored	

Theme Property Descriptions

int **check_v_offset** = 0

The vertical offset used when rendering the toggle icons (in pixels).

Texture2D **checked**

The icon to display when the **CheckButton** is checked (for left-to-right layouts).

Texture2D **checked_disabled** 🔗

The icon to display when the **CheckButton** is checked and disabled (for left-to-right layouts).

Texture2D **checked_disabled_mirrored** 🔗

The icon to display when the **CheckButton** is checked and disabled (for right-to-left layouts).

Texture2D **checked_mirrored** 🔗

The icon to display when the **CheckButton** is checked (for right-to-left layouts).

Texture2D **unchecked** 🔗

The icon to display when the **CheckButton** is unchecked (for left-to-right layouts).

Texture2D **unchecked_disabled** 🔗

The icon to display when the **CheckButton** is unchecked and disabled (for left-to-right layouts).

Texture2D **unchecked_disabled_mirrored** 🔗

The icon to display when the **CheckButton** is unchecked and disabled (for right-to-left layouts).

Texture2D **unchecked_mirrored** 🔗

The icon to display when the **CheckButton** is unchecked (for right-to-left layouts).

CodeEdit

Inherits: TextEdit **<** Control **<** CanvasItem **<** Node **<** Object

A multiline text editor designed for editing code.

Description

CodeEdit is a specialized TextEdit designed for editing plain text code files. It has many features commonly found in code editors such as line numbers, line folding, code completion, indent management, and string/comment management.

Note: Regardless of locale, **CodeEdit** will by default always use left-to-right text direction to correctly display source code.

Properties

bool	auto_brace_completion_enabled	`false`
bool	auto_brace_completion_highlight_matching	`false`
Dictionary	auto_brace_completion_pairs	`{ "\"": "\"", "'": "'", "(": ")", "[": "]", "{": "}" }`

bool	code_completion_enabled	`false`
Array[String]	code_completion_prefixes	`[]`
Array[String]	delimiter_comments	`[]`
Array[String]	delimiter_strings	`["' '", "\" \""]`
bool	gutters_draw_bookmarks	`false`
bool	gutters_draw_breakpoints_gutter	`false`
bool	gutters_draw_executing_lines	`false`
bool	gutters_draw_fold_gutter	`false`
bool	gutters_draw_line_numbers	`false`
bool	gutters_zero_pad_line_numbers	`false`
bool	indent_automatic	`false`
Array[String]	indent_automatic_prefixes	`[":", "{", "[", "("]`

int	indent_size	`4`
bool	indent_use_spaces	`false`
LayoutDirection	layout_direction	`2` (overrides Control)
bool	line_folding	`false`
Array[int]	line_length_guidelines	`[]`
bool	symbol_lookup_on_click	`false`
TextDirection	text_direction	`1` (overrides TextEdit)

Methods

void	_confirm_code_completion(replace: bool) virtual
Array[Dictionary]	_filter_code_completion_candidates(candidates: Array[Dictionary]) virtual const
void	_request_code_completion(force: bool) virtual

void	add_auto_brace_completion_pair(start_key: String, end_key: String)
void	add_code_completion_option(type: CodeCompletionKind, display_text: String, insert_text: String, text_color: Color = Color(1, 1, 1, 1), icon: Resource = null, value: Variant = null, location: int = 1024)
void	add_comment_delimiter(start_key: String, end_key: String, line_only: bool = false)
void	add_string_delimiter(start_key: String, end_key: String, line_only: bool = false)
bool	can_fold_line(line: int) const
void	cancel_code_completion()
void	clear_bookmarked_lines()
void	clear_breakpointed_lines()
void	clear_comment_delimiters()
void	clear_executing_lines()
void	clear_string_delimiters()

void	confirm_code_completion(replace: bool = false)
void	convert_indent(from_line: int = -1, to_line: int = -1)
void	create_code_region()
void	delete_lines()
void	do_indent()
void	duplicate_lines()
void	duplicate_selection()
void	fold_all_lines()
void	fold_line(line: int)
String	get_auto_brace_completion_close_key(open_key: String) const
PackedInt32Array	get_bookmarked_lines() const
PackedInt32Array	get_breakpointed_lines() const
Dictionary	get_code_completion_option(index: int) const

Array[Dictionary]	get_code_completion_options() const
int	get_code_completion_selected_index() const
String	get_code_region_end_tag() const
String	get_code_region_start_tag() const
String	get_delimiter_end_key(delimiter_index: int) const
Vector2	get_delimiter_end_position(line: int, column: int) const
String	get_delimiter_start_key(delimiter_index: int) const
Vector2	get_delimiter_start_position(line: int, column: int) const
PackedInt32Array	get_executing_lines() const
Array[int]	get_folded_lines() const
String	get_text_for_code_completion() const
String	get_text_for_symbol_lookup() const

String	get_text_with_cursor_char(line: int, column: int) const
bool	has_auto_brace_completion_close_key(close_key: String) const
bool	has_auto_brace_completion_open_key(open_key: String) const
bool	has_comment_delimiter(start_key: String) const
bool	has_string_delimiter(start_key: String) const
void	indent_lines()
int	is_in_comment(line: int, column: int = -1) const
int	is_in_string(line: int, column: int = -1) const
bool	is_line_bookmarked(line: int) const
bool	is_line_breakpointed(line: int) const
bool	is_line_code_region_end(line: int) const
bool	is_line_code_region_start(line: int) const

bool	is_line_executing(line: int) const
bool	is_line_folded(line: int) const
void	move_lines_down()
void	move_lines_up()
void	remove_comment_delimiter(start_key: String)
void	remove_string_delimiter(start_key: String)
void	request_code_completion(force: bool = false)
void	set_code_completion_selected_index(index: int)
void	set_code_hint(code_hint: String)
void	set_code_hint_draw_below(draw_below: bool)
void	set_code_region_tags(start: String = "region", end: String = "endregion")
void	set_line_as_bookmarked(line: int, bookmarked: bool)

void	set_line_as_breakpoint(line: int, breakpointed: bool)
void	set_line_as_executing(line: int, executing: bool)
void	set_symbol_lookup_word_as_valid(valid: bool)
void	toggle_foldable_line(line: int)
void	toggle_foldable_lines_at_carets()
void	unfold_all_lines()
void	unfold_line(line: int)
void	unindent_lines()
void	update_code_completion_options(force: bool)

Theme Properties

Color	bookmark_color	`Color(0.5, 0.64, 1, 0.8)`
Color	brace_mismatch_color	`Color(1, 0.2, 0.2, 1)`

Color	breakpoint_color	`Color(0.9, 0.29, 0.3, 1)`
Color	code_folding_color	`Color(0.8, 0.8, 0.8, 0.8)`
Color	completion_background_color	`Color(0.17, 0.16, 0.2, 1)`
Color	completion_existing_color	`Color(0.87, 0.87, 0.87, 0.13)`
Color	completion_scroll_color	`Color(1, 1, 1, 0.29)`
Color	completion_scroll_hovered_color	`Color(1, 1, 1, 0.4)`
Color	completion_selected_color	`Color(0.26, 0.26, 0.27, 1)`
Color	executing_line_color	`Color(0.98, 0.89, 0.27, 1)`
Color	folded_code_region_color	`Color(0.68, 0.46, 0.77, 0.2)`
Color	line_length_guideline_color	`Color(0.3, 0.5, 0.8, 0.1)`

Color	line_number_color	`Color(0.67, 0.67, 0.67, 0.4)`
int	completion_lines	`7`
int	completion_max_width	`50`
int	completion_scroll_width	`6`
Texture2D	bookmark	
Texture2D	breakpoint	
Texture2D	can_fold	
Texture2D	can_fold_code_region	
Texture2D	executing_line	
Texture2D	folded	
Texture2D	folded_code_region	
Texture2D	folded_eol_icon	
StyleBox	completion	

Signals

breakpoint_toggled(line: int)

Emitted when a breakpoint is added or removed from a line. If the line is moved via backspace a removed is emitted at the old line.

---

code_completion_requested()

Emitted when the user requests code completion.

---

symbol_lookup(symbol: String, line: int, column: int)

Emitted when the user has clicked on a valid symbol.

---

symbol_validate(symbol: String)

Emitted when the user hovers over a symbol. The symbol should be validated and responded to, by calling set_symbol_lookup_word_as_valid.

Enumerations

enum **CodeCompletionKind**:

CodeCompletionKind **KIND_CLASS** = `0`

Marks the option as a class.

CodeCompletionKind **KIND_FUNCTION** = `1`

Marks the option as a function.

CodeCompletionKind **KIND_SIGNAL** = `2`

Marks the option as a Godot signal.

CodeCompletionKind **KIND_VARIABLE** = `3`

Marks the option as a variable.

CodeCompletionKind **KIND_MEMBER** = `4`

Marks the option as a member.

CodeCompletionKind **KIND_ENUM** = `5`

Marks the option as an enum entry.

CodeCompletionKind **KIND_CONSTANT** = `6`

Marks the option as a constant.

CodeCompletionKind **KIND_NODE_PATH** = `7`

Marks the option as a Godot node path.

CodeCompletionKind **KIND_FILE_PATH** = `8`

Marks the option as a file path.

CodeCompletionKind **KIND_PLAIN_TEXT** = `9`

Marks the option as unclassified or plain text.

---

enum **CodeCompletionLocation**: 🔗

CodeCompletionLocation **LOCATION_LOCAL** = `0`

The option is local to the location of the code completion query - e.g. a local variable. Subsequent value of location represent options from the outer class, the exact value represent how far they are (in terms of inner classes).

CodeCompletionLocation **LOCATION_PARENT_MASK** = `256`

The option is from the containing class or a parent class, relative to the location of the code completion query. Perform a bitwise OR with the class depth (e.g. `0` for the local class, `1` for the parent, `2` for the grandparent, etc.) to store the depth of an option in the class or a parent class.

CodeCompletionLocation **LOCATION_OTHER_USER_CODE** = `512`

The option is from user code which is not local and not in a derived class (e.g. Autoload Singletons).

CodeCompletionLocation **LOCATION_OTHER** = `1024`

The option is from other engine code, not covered by the other enum constants - e.g. built-in classes.

Property Descriptions

bool **auto_brace_completion_enabled** = `false`

- void **set_auto_brace_completion_enabled**(value: bool)
- bool **is_auto_brace_completion_enabled**()

Sets whether brace pairs should be autocompleted.

---

bool **auto_brace_completion_highlight_matching** = `false`

- void **set_highlight_matching_braces_enabled**(value: bool)
- bool **is_highlight_matching_braces_enabled**()

Highlight mismatching brace pairs.

Dictionary **auto_brace_completion_pairs** = `{ "\"": "\"", "'": "'", "(": ")", "[": "]", "{": "}" }`

- void **set_auto_brace_completion_pairs**(value: Dictionary)
- Dictionary **get_auto_brace_completion_pairs**()

Sets the brace pairs to be autocompleted.

---

bool **code_completion_enabled** = `false`

- void **set_code_completion_enabled**(value: bool)
- bool **is_code_completion_enabled**()

Sets whether code completion is allowed.

Array[String] **code_completion_prefixes** = `[]`

- void **set_code_completion_prefixes**(value: Array[String])
- Array[String] **get_code_completion_prefixes**()

Sets prefixes that will trigger code completion.

Array[String] **delimiter_comments** = `[]`

- void **set_comment_delimiters**(value: Array[String])
- Array[String] **get_comment_delimiters**()

Sets the comment delimiters. All existing comment delimiters will be removed.

Array[String] **delimiter_strings** = `["' '", "\" \""]`

- void **set_string_delimiters**(value: Array[String])
- Array[String] **get_string_delimiters**()

Sets the string delimiters. All existing string delimiters will be removed.

---

bool **gutters_draw_bookmarks** = `false`

- void **set_draw_bookmarks_gutter**(value: bool)
- bool **is_drawing_bookmarks_gutter**()

Sets if bookmarked should be drawn in the gutter. This gutter is shared with breakpoints and executing lines.

---

bool **gutters_draw_breakpoints_gutter** = `false`

- void **set_draw_breakpoints_gutter**(value: bool)
- bool **is_drawing_breakpoints_gutter**()

Sets if breakpoints should be drawn in the gutter. This gutter is shared with bookmarks and executing lines.

---

bool **gutters_draw_executing_lines** = `false`

- void **set_draw_executing_lines_gutter**(value: bool)
- bool **is_drawing_executing_lines_gutter**()

Sets if executing lines should be marked in the gutter. This gutter is shared with breakpoints and bookmarks lines.

---

bool **gutters_draw_fold_gutter** = `false`

- void **set_draw_fold_gutter**(value: bool)
- bool **is_drawing_fold_gutter**()

Sets if foldable lines icons should be drawn in the gutter.

---

bool **gutters_draw_line_numbers** = `false` 🔗

- void **set_draw_line_numbers**(value: bool)
- bool **is_draw_line_numbers_enabled**()

Sets if line numbers should be drawn in the gutter.

---

bool **gutters_zero_pad_line_numbers** = `false` 🔗

- void **set_line_numbers_zero_padded**(value: bool)
- bool **is_line_numbers_zero_padded**()

Sets if line numbers drawn in the gutter are zero padded.

---

bool **indent_automatic** = `false` 🔗

- void **set_auto_indent_enabled**(value: bool)
- bool **is_auto_indent_enabled**()

Sets whether automatic indent are enabled, this will add an extra indent if a prefix or brace is found.

Array[String] **indent_automatic_prefixes** = `[":", "{", "[", "("]` 🔗

- void **set_auto_indent_prefixes**(value: Array[String])
- Array[String] **get_auto_indent_prefixes**()

Prefixes to trigger an automatic indent.

---

int **indent_size** = `4` 🔗

- void **set_indent_size**(value: int)
- int **get_indent_size**()

Size of the tabulation indent (one `Tab` press) in characters. If indent_use_spaces is enabled the number of spaces to use.

---

bool **indent_use_spaces** = `false`

- void **set_indent_using_spaces**(value: bool)
- bool **is_indent_using_spaces**()

Use spaces instead of tabs for indentation.

---

bool **line_folding** = `false`

- void **set_line_folding_enabled**(value: bool)
- bool **is_line_folding_enabled**()

Sets whether line folding is allowed.

Array[int] **line_length_guidelines** = `[]`

- void **set_line_length_guidelines**(value: Array[int])
- Array[int] **get_line_length_guidelines**()

Draws vertical lines at the provided columns. The first entry is considered a main hard guideline and is draw more prominently.

---

bool **symbol_lookup_on_click** = `false`

- void **set_symbol_lookup_on_click_enabled**(value: bool)
- bool **is_symbol_lookup_on_click_enabled**()

Set when a validated word from symbol_validate is clicked, the symbol_lookup should be emitted.

Method Descriptions

void **_confirm_code_completion**(replace: bool) virtual

Override this method to define how the selected entry should be inserted. If `replace` is `true`, any existing text should be replaced.

Array[Dictionary] **_filter_code_completion_candidates**(candidates: Array[Dictionary]) virtual const

Override this method to define what items in `candidates` should be displayed.

Both `candidates` and the return is a Array of Dictionary, see get_code_completion_option for Dictionary content.

---

void **_request_code_completion**(force: bool) virtual

Override this method to define what happens when the user requests code completion. If `force` is `true`, any checks should be bypassed.

---

void **add_auto_brace_completion_pair**(start_key: String, end_key: String)

Adds a brace pair.

Both the start and end keys must be symbols. Only the start key has to be unique.

---

void **add_code_completion_option**(type: CodeCompletionKind, display_text: String, insert_text: String, text_color: Color = Color(1, 1, 1, 1), icon: Resource = null, value: Variant = null, location: int = 1024)

Submits an item to the queue of potential candidates for the autocomplete menu. Call update_code_completion_options to update the list.

`location` indicates location of the option relative to the location of the code completion query. See CodeCompletionLocation for how to set this value.

Note: This list will replace all current candidates.

---

void **add_comment_delimiter**(start_key: String, end_key: String, line_only: bool = false)

Adds a comment delimiter from `start_key` to `end_key`. Both keys should be symbols, and `start_key` must not be shared with other delimiters.

If `line_only` is `true` or `end_key` is an empty String, the region does not carry over to the next line.

---

void **add_string_delimiter**(start_key: String, end_key: String, line_only: bool = false)

Defines a string delimiter from `start_key` to `end_key`. Both keys should be symbols, and `start_key` must not be shared with other delimiters.

If `line_only` is `true` or `end_key` is an empty String, the region does not carry over to the next line.

---

bool **can_fold_line**(line: int) const

Returns if the given line is foldable, that is, it has indented lines right below it or a comment / string block.

---

void **cancel_code_completion**()

Cancels the autocomplete menu.

void **clear_bookmarked_lines**()

Clears all bookmarked lines.

void **clear_breakpointed_lines**()

Clears all breakpointed lines.

void **clear_comment_delimiters**()

Removes all comment delimiters.

void **clear_executing_lines**()

Clears all executed lines.

void **clear_string_delimiters**()

Removes all string delimiters.

void **confirm_code_completion**(replace: bool = false)

Inserts the selected entry into the text. If `replace` is `true`, any existing text is replaced rather than merged.

void **convert_indent**(from_line: int = -1, to_line: int = -1)

Converts the indents of lines between `from_line` and `to_line` to tabs or spaces as set by indent_use_spaces.

Values of `-1` convert the entire text.

---

void **create_code_region**()

Creates a new code region with the selection. At least one single line comment delimiter have to be defined (see add_comment_delimiter).

A code region is a part of code that is highlighted when folded and can help organize your script.

Code region start and end tags can be customized (see set_code_region_tags).

Code regions are delimited using start and end tags (respectively `region` and `endregion` by default) preceded by one line comment delimiter. (eg. `#region` and `#endregion`)

---

void **delete_lines**()

Deletes all lines that are selected or have a caret on them.

---

void **do_indent**()

Perform an indent as if the user activated the "ui_text_indent" action.

---

void **duplicate_lines**()

Duplicates all lines currently selected with any caret. Duplicates the entire line beneath the current one no matter where the caret is within the line.

---

void **duplicate_selection**()

Duplicates all selected text and duplicates all lines with a caret on them.

---

void **fold_all_lines**()

Folds all lines that are possible to be folded (see can_fold_line).

---

void **fold_line**(line: int)

Folds the given line, if possible (see can_fold_line).

String **get_auto_brace_completion_close_key**(open_key: String) const

Gets the matching auto brace close key for `open_key`.

PackedInt32Array **get_bookmarked_lines**() const

Gets all bookmarked lines.

PackedInt32Array **get_breakpointed_lines**() const

Gets all breakpointed lines.

Dictionary **get_code_completion_option**(index: int) const

Gets the completion option at `index`. The return Dictionary has the following key-values:

`kind`: CodeCompletionKind

`display_text`: Text that is shown on the autocomplete menu.

`insert_text`: Text that is to be inserted when this item is selected.

`font_color`: Color of the text on the autocomplete menu.

`icon`: Icon to draw on the autocomplete menu.

`default_value`: Value of the symbol.

Array[Dictionary] **get_code_completion_options**() const

Gets all completion options, see get_code_completion_option for return content.

---

int **get_code_completion_selected_index**() const

Gets the index of the current selected completion option.

String **get_code_region_end_tag**() const

Returns the code region end tag (without comment delimiter).

String **get_code_region_start_tag**() const

Returns the code region start tag (without comment delimiter).

String **get_delimiter_end_key**(delimiter_index: int) const

Gets the end key for a string or comment region index.

Vector2 **get_delimiter_end_position**(line: int, column: int) const

If `line` `column` is in a string or comment, returns the end position of the region. If not or no end could be found, both Vector2 values will be `-1`.

String **get_delimiter_start_key**(delimiter_index: int) const

Gets the start key for a string or comment region index.

Vector2 **get_delimiter_start_position**(line: int, column: int) const

If `line` `column` is in a string or comment, returns the start position of the region. If not or no start could be found, both Vector2 values will be `-1`.

PackedInt32Array **get_executing_lines**() const

Gets all executing lines.

Array[int] **get_folded_lines**() const

Returns all lines that are current folded.

String **get_text_for_code_completion**() const

Returns the full text with char `0xFFFF` at the caret location.

String **get_text_for_symbol_lookup**() const

Returns the full text with char `0xFFFF` at the cursor location.

String **get_text_with_cursor_char**(line: int, column: int) const

Returns the full text with char `0xFFFF` at the specified location.

---

bool **has_auto_brace_completion_close_key**(close_key: String) const

Returns `true` if close key `close_key` exists.

---

bool **has_auto_brace_completion_open_key**(open_key: String) const 🔗

Returns `true` if open key `open_key` exists.

---

bool **has_comment_delimiter**(start_key: String) const 🔗

Returns `true` if comment `start_key` exists.

---

bool **has_string_delimiter**(start_key: String) const 🔗

Returns `true` if string `start_key` exists.

---

void **indent_lines**() 🔗

Indents selected lines, or in the case of no selection the caret line by one.

---

int **is_in_comment**(line: int, column: int = -1) const 🔗

Returns delimiter index if `line` `column` is in a comment. If `column` is not provided, will return delimiter index if the entire `line` is a comment. Otherwise `-1`.

---

int **is_in_string**(line: int, column: int = -1) const 🔗

Returns the delimiter index if `line` `column` is in a string. If `column` is not provided, will return the delimiter index if the entire `line` is a string. Otherwise `-1`.

---

bool **is_line_bookmarked**(line: int) const 🔗

Returns whether the line at the specified index is bookmarked or not.

bool **is_line_breakpointed**(line: int) const

Returns whether the line at the specified index is breakpointed or not.

bool **is_line_code_region_end**(line: int) const

Returns whether the line at the specified index is a code region end.

bool **is_line_code_region_start**(line: int) const

Returns whether the line at the specified index is a code region start.

bool **is_line_executing**(line: int) const

Returns whether the line at the specified index is marked as executing or not.

bool **is_line_folded**(line: int) const

Returns whether the line at the specified index is folded or not.

void **move_lines_down**()

Moves all lines down that are selected or have a caret on them.

void **move_lines_up**()

Moves all lines up that are selected or have a caret on them.

void **remove_comment_delimiter**(start_key: String) 🔗

Removes the comment delimiter with `start_key`.

---

void **remove_string_delimiter**(start_key: String) 🔗

Removes the string delimiter with `start_key`.

---

void **request_code_completion**(force: bool = false) 🔗

Emits code_completion_requested, if `force` is `true` will bypass all checks. Otherwise will check that the caret is in a word or in front of a prefix. Will ignore the request if all current options are of type file path, node path, or signal.

---

void **set_code_completion_selected_index**(index: int) 🔗

Sets the current selected completion option.

---

void **set_code_hint**(code_hint: String) 🔗

Sets the code hint text. Pass an empty string to clear.

---

void **set_code_hint_draw_below**(draw_below: bool) 🔗

Sets if the code hint should draw below the text.

---

void **set_code_region_tags**(start: String = "region", end: String = "endregion") 🔗

Sets the code region start and end tags (without comment delimiter).

---

void **set_line_as_bookmarked**(line: int, bookmarked: bool) 🔗

Sets the line as bookmarked.

---

void **set_line_as_breakpoint**(line: int, breakpointed: bool) 🔗

Sets the line as breakpointed.

---

void **set_line_as_executing**(line: int, executing: bool) 🔗

Sets the line as executing.

---

void **set_symbol_lookup_word_as_valid**(valid: bool) 🔗

Sets the symbol emitted by symbol_validate as a valid lookup.

---

void **toggle_foldable_line**(line: int) 🔗

Toggle the folding of the code block at the given line.

---

void **toggle_foldable_lines_at_carets**() 🔗

Toggle the folding of the code block on all lines with a caret on them.

---

void **unfold_all_lines**() 🔗

Unfolds all lines, folded or not.

---

void **unfold_line**(line: int) 🔗

Unfolds all lines that were previously folded.

---

void **unindent_lines**()

Unindents selected lines, or in the case of no selection the caret line by one. Same as performing "ui_text_unindent" action.

---

void **update_code_completion_options**(force: bool)

Submits all completion options added with add_code_completion_option. Will try to force the autocomplete menu to popup, if `force` is `true`.

Note: This will replace all current candidates.

Theme Property Descriptions

Color **bookmark_color** = `Color(0.5, 0.64, 1, 0.8)`

Color of the bookmark icon for bookmarked lines.

Color **brace_mismatch_color** = `Color(1, 0.2, 0.2, 1)`

Color of the text to highlight mismatched braces.

Color **breakpoint_color** = `Color(0.9, 0.29, 0.3, 1)`

Color of the breakpoint icon for bookmarked lines.

Color **code_folding_color** = `Color(0.8, 0.8, 0.8, 0.8)`

Color for all icons related to line folding.

Color **completion_background_color** = `Color(0.17, 0.16, 0.2, 1)`

Sets the background Color for the code completion popup.

Color **completion_existing_color** = `Color(0.87, 0.87, 0.87, 0.13)`

Background highlight Color for matching text in code completion options.

Color **completion_scroll_color** = `Color(1, 1, 1, 0.29)`

Color of the scrollbar in the code completion popup.

Color **completion_scroll_hovered_color** = `Color(1, 1, 1, 0.4)`

Color of the scrollbar in the code completion popup when hovered.

Color **completion_selected_color** = `Color(0.26, 0.26, 0.27, 1)`

Background highlight Color for the current selected option item in the code completion popup.

Color **executing_line_color** = `Color(0.98, 0.89, 0.27, 1)`

Color of the executing icon for executing lines.

Color **folded_code_region_color** = `Color(0.68, 0.46, 0.77, 0.2)`

Color of background line highlight for folded code region.

Color **line_length_guideline_color** = `Color(0.3, 0.5, 0.8, 0.1)`

Color of the main line length guideline, secondary guidelines will have 50% alpha applied.

Color **line_number_color** = `Color(0.67, 0.67, 0.67, 0.4)`

Sets the Color of line numbers.

int **completion_lines** = `7`

Max number of options to display in the code completion popup at any one time.

int **completion_max_width** = `50`

Max width of options in the code completion popup. Options longer than this will be cut off.

int **completion_scroll_width** = `6`

Width of the scrollbar in the code completion popup.

Texture2D **bookmark**

Sets a custom Texture2D to draw in the bookmark gutter for bookmarked lines.

Texture2D **breakpoint**

Sets a custom Texture2D to draw in the breakpoint gutter for breakpointed lines.

Texture2D **can_fold**

Sets a custom Texture2D to draw in the line folding gutter when a line can be folded.

Texture2D **can_fold_code_region**

Sets a custom Texture2D to draw in the line folding gutter when a code region can be folded.

Texture2D **executing_line**

Icon to draw in the executing gutter for executing lines.

Texture2D **folded**

Sets a custom Texture2D to draw in the line folding gutter when a line is folded and can be unfolded.

Texture2D **folded_code_region**

Sets a custom Texture2D to draw in the line folding gutter when a code region is folded and can be unfolded.

Texture2D **folded_eol_icon**

Sets a custom Texture2D to draw at the end of a folded line.

StyleBox **completion**

StyleBox for the code completion popup.

CollisionObject2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Inherited By: Area2D, PhysicsBody2D

Abstract base class for 2D physics objects.

Description

Abstract base class for 2D physics objects. **CollisionObject2D** can hold any number of Shape2Ds for collision. Each shape must be assigned to a *shape owner*. Shape owners are not nodes and do not appear in the editor, but are accessible through code using the `shape_owner_*` methods.

Note: Only collisions between objects within the same canvas (Viewport canvas or CanvasLayer) are supported. The behavior of collisions between objects in different canvases is undefined.

Properties

int	collision_layer	`1`
int	collision_mask	`1`
float	collision_priority	`1.0`
DisableMode	disable_mode	`0`
bool	input_pickable	`true`

Methods

void	_input_event(viewport: Viewport, event: InputEvent, shape_idx: int) virtual
void	_mouse_enter() virtual
void	_mouse_exit() virtual
void	_mouse_shape_enter(shape_idx: int) virtual
void	_mouse_shape_exit(shape_idx: int) virtual
int	create_shape_owner(owner: Object)
bool	get_collision_layer_value(layer_number: int) const
bool	get_collision_mask_value(layer_number: int) const
RID	get_rid() const
float	get_shape_owner_one_way_collision_margin(owner_id: int) const
PackedInt32Array	get_shape_owners()
bool	is_shape_owner_disabled(owner_id: int) const

bool	is_shape_owner_one_way_collision_enabled(owner_id: int) const
void	remove_shape_owner(owner_id: int)
void	set_collision_layer_value(layer_number: int, value: bool)
void	set_collision_mask_value(layer_number: int, value: bool)
int	shape_find_owner(shape_index: int) const
void	shape_owner_add_shape(owner_id: int, shape: Shape2D)
void	shape_owner_clear_shapes(owner_id: int)
Object	shape_owner_get_owner(owner_id: int) const
Shape2D	shape_owner_get_shape(owner_id: int, shape_id: int) const
int	shape_owner_get_shape_count(owner_id: int) const
int	shape_owner_get_shape_index(owner_id: int, shape_id: int) const
Transform2D	shape_owner_get_transform(owner_id: int) const

void	shape_owner_remove_shape(owner_id: int, shape_id: int)
void	shape_owner_set_disabled(owner_id: int, disabled: bool)
void	shape_owner_set_one_way_collision(owner_id: int, enable: bool)
void	shape_owner_set_one_way_collision_margin(owner_id: int, margin: float)
void	shape_owner_set_transform(owner_id: int, transform: Transform2D)

Signals

input_event(viewport: Node, event: InputEvent, shape_idx: int) 🔗

Emitted when an input event occurs. Requires input_pickable to be `true` and at least one collision_layer bit to be set. See _input_event for details.

---

mouse_entered() 🔗

Emitted when the mouse pointer enters any of this object's shapes. Requires input_pickable to be `true` and at least one collision_layer bit to be set. Note that moving between different shapes within a single **CollisionObject2D** won't cause this signal to be emitted.

Note: Due to the lack of continuous collision detection, this signal may not be emitted in the expected order if the mouse moves fast enough and the **CollisionObject2D**'s area is small. This signal may also not be emitted if another **CollisionObject2D** is overlapping the **CollisionObject2D** in question.

---

mouse_exited() 🔗

Emitted when the mouse pointer exits all this object's shapes. Requires input_pickable to be `true` and at least one collision_layer bit to be set. Note that moving between different shapes within a single **CollisionObject2D** won't cause this signal to be emitted.

Note: Due to the lack of continuous collision detection, this signal may not be emitted in the expected order if the mouse moves fast enough and the **CollisionObject2D**'s area is small. This signal may also not be emitted if another **CollisionObject2D** is overlapping the **CollisionObject2D** in question.

---

mouse_shape_entered(shape_idx: int) 🔗

Emitted when the mouse pointer enters any of this object's shapes or moves from one shape to another. `shape_idx` is the child index of the newly entered Shape2D. Requires input_pickable to be `true` and at least one collision_layer bit to be set.

---

mouse_shape_exited(shape_idx: int) 🔗

Emitted when the mouse pointer exits any of this object's shapes. `shape_idx` is the child index of the exited Shape2D. Requires input_pickable to be `true` and at least one collision_layer bit to be set.

Enumerations

enum **DisableMode**: 🔗

DisableMode **DISABLE_MODE_REMOVE** = `0`

When Node.process_mode is set to Node.PROCESS_MODE_DISABLED, remove from the physics simulation to stop all physics interactions with this **CollisionObject2D**.

Automatically re-added to the physics simulation when the Node is processed again.

DisableMode **DISABLE_MODE_MAKE_STATIC** = `1`

When Node.process_mode is set to Node.PROCESS_MODE_DISABLED, make the body static. Doesn't affect Area2D. PhysicsBody2D can't be affected by forces or other bodies while static.

Automatically set PhysicsBody2D back to its original mode when the Node is processed again.

DisableMode **DISABLE_MODE_KEEP_ACTIVE** = `2`

When Node.process_mode is set to Node.PROCESS_MODE_DISABLED, do not affect the physics simulation.

Property Descriptions

int **collision_layer** = `1` 🔗

- void **set_collision_layer**(value: int)
- int **get_collision_layer**()

The physics layers this CollisionObject2D is in. Collision objects can exist in one or more of 32 different layers. See also collision_mask.

Note: Object A can detect a contact with object B only if object B is in any of the layers that object A scans. See Collision layers and masks in the documentation for more information.

int **collision_mask** = `1` 🔗

- void **set_collision_mask**(value: int)

- int **get_collision_mask**()

The physics layers this CollisionObject2D scans. Collision objects can scan one or more of 32 different layers. See also collision_layer.

Note: Object A can detect a contact with object B only if object B is in any of the layers that object A scans. See Collision layers and masks in the documentation for more information.

---

float **collision_priority** = `1.0`

- void **set_collision_priority**(value: float)
- float **get_collision_priority**()

The priority used to solve colliding when occurring penetration. The higher the priority is, the lower the penetration into the object will be. This can for example be used to prevent the player from breaking through the boundaries of a level.

DisableMode **disable_mode** = `0`

- void **set_disable_mode**(value: DisableMode)
- DisableMode **get_disable_mode**()

Defines the behavior in physics when Node.process_mode is set to Node.PROCESS_MODE_DISABLED. See DisableMode for more details about the different modes.

---

bool **input_pickable** = `true`

- void **set_pickable**(value: bool)
- bool **is_pickable**()

If `true`, this object is pickable. A pickable object can detect the mouse pointer entering/leaving, and if the mouse is inside it, report input events. Requires at least one collision_layer bit to be set.

Method Descriptions

void **_input_event**(viewport: Viewport, event: InputEvent, shape_idx: int) virtual 🔗

Accepts unhandled InputEvents. `shape_idx` is the child index of the clicked Shape2D. Connect to input_event to easily pick up these events.

Note: _input_event requires input_pickable to be `true` and at least one collision_layer bit to be set.

---

void **_mouse_enter**() virtual 🔗

Called when the mouse pointer enters any of this object's shapes. Requires input_pickable to be `true` and at least one collision_layer bit to be set. Note that moving between different shapes within a single **CollisionObject2D** won't cause this function to be called.

---

void **_mouse_exit**() virtual 🔗

Called when the mouse pointer exits all this object's shapes. Requires input_pickable to be `true` and at least one collision_layer bit to be set. Note that moving between different shapes within a single **CollisionObject2D** won't cause this function to be called.

---

void **_mouse_shape_enter**(shape_idx: int) virtual 🔗

Called when the mouse pointer enters any of this object's shapes or moves from one shape to another. `shape_idx` is the child index of the newly entered Shape2D. Requires input_pickable to be `true` and at least one collision_layer bit to be called.

---

void **_mouse_shape_exit**(shape_idx: int) virtual 🔗

Called when the mouse pointer exits any of this object's shapes. `shape_idx` is the child index of the exited Shape2D. Requires input_pickable to be `true` and at least one collision_layer bit to be called.

---

int **create_shape_owner**(owner: Object)

Creates a new shape owner for the given object. Returns `owner_id` of the new owner for future reference.

---

bool **get_collision_layer_value**(layer_number: int) const

Returns whether or not the specified layer of the collision_layer is enabled, given a `layer_number` between 1 and 32.

---

bool **get_collision_mask_value**(layer_number: int) const

Returns whether or not the specified layer of the collision_mask is enabled, given a `layer_number` between 1 and 32.

RID **get_rid**() const

Returns the object's RID.

---

float **get_shape_owner_one_way_collision_margin**(owner_id: int) const

Returns the `one_way_collision_margin` of the shape owner identified by given `owner_id`.

PackedInt32Array **get_shape_owners**()

Returns an Array of `owner_id` identifiers. You can use these ids in other methods that take `owner_id` as an argument.

---

bool **is_shape_owner_disabled**(owner_id: int) const

If `true`, the shape owner and its shapes are disabled.

---

bool **is_shape_owner_one_way_collision_enabled**(owner_id: int) const

Returns `true` if collisions for the shape owner originating from this **CollisionObject2D** will not be reported to collided with **CollisionObject2D**s.

---

void **remove_shape_owner**(owner_id: int)

Removes the given shape owner.

---

void **set_collision_layer_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the collision_layer, given a `layer_number` between 1 and 32.

---

void **set_collision_mask_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the collision_mask, given a `layer_number` between 1 and 32.

---

int **shape_find_owner**(shape_index: int) const

Returns the `owner_id` of the given shape.

---

void **shape_owner_add_shape**(owner_id: int, shape: Shape2D)

Adds a Shape2D to the shape owner.

---

void **shape_owner_clear_shapes**(owner_id: int) 🔗

Removes all shapes from the shape owner.

Object **shape_owner_get_owner**(owner_id: int) const 🔗

Returns the parent object of the given shape owner.

Shape2D **shape_owner_get_shape**(owner_id: int, shape_id: int) const 🔗

Returns the Shape2D with the given ID from the given shape owner.

---

int **shape_owner_get_shape_count**(owner_id: int) const 🔗

Returns the number of shapes the given shape owner contains.

---

int **shape_owner_get_shape_index**(owner_id: int, shape_id: int) const 🔗

Returns the child index of the Shape2D with the given ID from the given shape owner.

Transform2D **shape_owner_get_transform**(owner_id: int) const 🔗

Returns the shape owner's Transform2D.

---

void **shape_owner_remove_shape**(owner_id: int, shape_id: int) 🔗

Removes a shape from the given shape owner.

---

void **shape_owner_set_disabled**(owner_id: int, disabled: bool) 🔗

If `true`, disables the given shape owner.

---

void **shape_owner_set_one_way_collision**(owner_id: int, enable: bool)

If `enable` is `true`, collisions for the shape owner originating from this **CollisionObject2D** will not be reported to collided with **CollisionObject2D**s.

---

void **shape_owner_set_one_way_collision_margin**(owner_id: int, margin: float)

Sets the `one_way_collision_margin` of the shape owner identified by given `owner_id` to `margin` pixels.

---

void **shape_owner_set_transform**(owner_id: int, transform: Transform2D)

Sets the Transform2D of the given shape owner.

CollisionObject3D

Inherits: Node3D **<** Node **<** Object

Inherited By: Area3D, PhysicsBody3D

Abstract base class for 3D physics objects.

Description

Abstract base class for 3D physics objects. **CollisionObject3D** can hold any number of Shape3Ds for collision. Each shape must be assigned to a *shape owner*. Shape owners are not nodes and do not appear in the editor, but are accessible through code using the `shape_owner_*` methods.

Warning: With a non-uniform scale, this node will likely not behave as expected. It is advised to keep its scale the same on all axes and adjust its collision shape(s) instead.

Properties

int	collision_layer	`1`
int	collision_mask	`1`
float	collision_priority	`1.0`

DisableMode	disable_mode	`0`
bool	input_capture_on_drag	`false`
bool	input_ray_pickable	`true`

Methods

void	_input_event(camera: Camera3D, event: InputEvent, event_position: Vector3, normal: Vector3, shape_idx: int) virtual
void	_mouse_enter() virtual
void	_mouse_exit() virtual
int	create_shape_owner(owner: Object)
bool	get_collision_layer_value(layer_number: int) const
bool	get_collision_mask_value(layer_number: int) const

RID	get_rid() const
PackedInt32Array	get_shape_owners()
bool	is_shape_owner_disabled(owner_id: int) const
void	remove_shape_owner(owner_id: int)
void	set_collision_layer_value(layer_number: int, value: bool)
void	set_collision_mask_value(layer_number: int, value: bool)
int	shape_find_owner(shape_index: int) const
void	shape_owner_add_shape(owner_id: int, shape: Shape3D)
void	shape_owner_clear_shapes(owner_id: int)
Object	shape_owner_get_owner(owner_id: int) const
Shape3D	shape_owner_get_shape(owner_id: int, shape_id: int) const

int	shape_owner_get_shape_count(owner_id: int) const
int	shape_owner_get_shape_index(owner_id: int, shape_id: int) const
Transform3D	shape_owner_get_transform(owner_id: int) const
void	shape_owner_remove_shape(owner_id: int, shape_id: int)
void	shape_owner_set_disabled(owner_id: int, disabled: bool)
void	shape_owner_set_transform(owner_id: int, transform: Transform3D)

Signals

input_event(camera: Node, event: InputEvent, event_position: Vector3, normal: Vector3, shape_idx: int) 🔗

Emitted when the object receives an unhandled InputEvent. `event_position` is the location in world space of the mouse pointer on the surface of the shape with index `shape_idx` and `normal` is the normal vector of the surface at that point.

mouse_entered() 🔗

Emitted when the mouse pointer enters any of this object's shapes. Requires input_ray_pickable to be `true` and at least one collision_layer bit to be set.

Note: Due to the lack of continuous collision detection, this signal may not be emitted in the expected order if the mouse moves fast enough and the **CollisionObject3D**'s area is small. This signal may also not be emitted if another **CollisionObject3D** is overlapping the **CollisionObject3D** in question.

---

mouse_exited() 🔗

Emitted when the mouse pointer exits all this object's shapes. Requires input_ray_pickable to be `true` and at least one collision_layer bit to be set.

Note: Due to the lack of continuous collision detection, this signal may not be emitted in the expected order if the mouse moves fast enough and the **CollisionObject3D**'s area is small. This signal may also not be emitted if another **CollisionObject3D** is overlapping the **CollisionObject3D** in question.

Enumerations

enum **DisableMode**: 🔗

DisableMode **DISABLE_MODE_REMOVE** = `0`

When Node.process_mode is set to Node.PROCESS_MODE_DISABLED, remove from the physics simulation to stop all physics interactions with this **CollisionObject3D**.

Automatically re-added to the physics simulation when the Node is processed again.

DisableMode **DISABLE_MODE_MAKE_STATIC** = `1`

When Node.process_mode is set to Node.PROCESS_MODE_DISABLED, make the body static. Doesn't affect Area3D. PhysicsBody3D can't be affected by forces or other bodies while static.

Automatically set PhysicsBody3D back to its original mode when the Node is processed again.

DisableMode **DISABLE_MODE_KEEP_ACTIVE** = `2`

When Node.process_mode is set to Node.PROCESS_MODE_DISABLED, do not affect the physics simulation.

Property Descriptions

int **collision_layer** = `1`

- void **set_collision_layer**(value: int)
- int **get_collision_layer**()

The physics layers this CollisionObject3D **is in**. Collision objects can exist in one or more of 32 different layers. See also collision_mask.

Note: Object A can detect a contact with object B only if object B is in any of the layers that object A scans. See Collision layers and masks in the documentation for more information.

int **collision_mask** = `1`

- void **set_collision_mask**(value: int)
- int **get_collision_mask**()

The physics layers this CollisionObject3D **scans**. Collision objects can scan one or more of 32 different layers. See also collision_layer.

Note: Object A can detect a contact with object B only if object B is in any of the layers that object A scans. See Collision layers and masks in the documentation for more information.

---

float **collision_priority** = `1.0` 🔗

- void **set_collision_priority**(value: float)
- float **get_collision_priority**()

The priority used to solve colliding when occurring penetration. The higher the priority is, the lower the penetration into the object will be. This can for example be used to prevent the player from breaking through the boundaries of a level.

DisableMode **disable_mode** = `0` 🔗

- void **set_disable_mode**(value: DisableMode)
- DisableMode **get_disable_mode**()

Defines the behavior in physics when Node.process_mode is set to Node.PROCESS_MODE_DISABLED. See DisableMode for more details about the different modes.

---

bool **input_capture_on_drag** = `false` 🔗

- void **set_capture_input_on_drag**(value: bool)
- bool **get_capture_input_on_drag**()

If `true`, the **CollisionObject3D** will continue to receive input events as the mouse is dragged across its shapes.

---

bool **input_ray_pickable** = `true` 🔗

- void **set_ray_pickable**(value: bool)
- bool **is_ray_pickable**()

If `true`, this object is pickable. A pickable object can detect the mouse pointer entering/leaving, and if the mouse is inside it, report input events. Requires at least one collision_layer bit to be set.

Method Descriptions

void **_input_event**(camera: Camera3D, event: InputEvent, event_position: Vector3, normal: Vector3, shape_idx: int) virtual 🔗

Receives unhandled InputEvents. `event_position` is the location in world space of the mouse pointer on the surface of the shape with index `shape_idx` and `normal` is the normal vector of the surface at that point. Connect to the input_event signal to easily pick up these events.

Note: _input_event requires input_ray_pickable to be `true` and at least one collision_layer bit to be set.

---

void **_mouse_enter**() virtual 🔗

Called when the mouse pointer enters any of this object's shapes. Requires input_ray_pickable to be `true` and at least one collision_layer bit to be set. Note that moving between different shapes within a single **CollisionObject3D** won't cause this function to be called.

---

void **_mouse_exit**() virtual

Called when the mouse pointer exits all this object's shapes. Requires input_ray_pickable to be `true` and at least one collision_layer bit to be set. Note that moving between different shapes within a single **CollisionObject3D** won't cause this function to be called.

---

int **create_shape_owner**(owner: Object)

Creates a new shape owner for the given object. Returns `owner_id` of the new owner for future reference.

---

bool **get_collision_layer_value**(layer_number: int) const

Returns whether or not the specified layer of the collision_layer is enabled, given a `layer_number` between 1 and 32.

---

bool **get_collision_mask_value**(layer_number: int) const

Returns whether or not the specified layer of the collision_mask is enabled, given a `layer_number` between 1 and 32.

RID **get_rid**() const

Returns the object's RID.

PackedInt32Array **get_shape_owners**() 🔗

Returns an Array of `owner_id` identifiers. You can use these ids in other methods that take `owner_id` as an argument.

---

bool **is_shape_owner_disabled**(owner_id: int) const 🔗

If `true`, the shape owner and its shapes are disabled.

---

void **remove_shape_owner**(owner_id: int) 🔗

Removes the given shape owner.

---

void **set_collision_layer_value**(layer_number: int, value: bool) 🔗

Based on `value`, enables or disables the specified layer in the collision_layer, given a `layer_number` between 1 and 32.

---

void **set_collision_mask_value**(layer_number: int, value: bool) 🔗

Based on `value`, enables or disables the specified layer in the collision_mask, given a `layer_number` between 1 and 32.

---

int **shape_find_owner**(shape_index: int) const 🔗

Returns the `owner_id` of the given shape.

---

void **shape_owner_add_shape**(owner_id: int, shape: Shape3D)

Adds a Shape3D to the shape owner.

---

void **shape_owner_clear_shapes**(owner_id: int)

Removes all shapes from the shape owner.

Object **shape_owner_get_owner**(owner_id: int) const

Returns the parent object of the given shape owner.

Shape3D **shape_owner_get_shape**(owner_id: int, shape_id: int) const

Returns the Shape3D with the given ID from the given shape owner.

---

int **shape_owner_get_shape_count**(owner_id: int) const

Returns the number of shapes the given shape owner contains.

---

int **shape_owner_get_shape_index**(owner_id: int, shape_id: int) const

Returns the child index of the Shape3D with the given ID from the given shape owner.

Transform3D **shape_owner_get_transform**(owner_id: int) const

Returns the shape owner's Transform3D.

---

void **shape_owner_remove_shape**(owner_id: int, shape_id: int)

Removes a shape from the given shape owner.

---

void **shape_owner_set_disabled**(owner_id: int, disabled: bool)

If `true`, disables the given shape owner.

---

void **shape_owner_set_transform**(owner_id: int, transform: Transform3D)

Sets the Transform3D of the given shape owner.

CollisionPolygon2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A node that provides a polygon shape to a CollisionObject2D parent.

Description

A node that provides a polygon shape to a CollisionObject2D parent and allows to edit it. The polygon can be concave or convex. This can give a detection shape to an Area2D, turn PhysicsBody2D into a solid object, or give a hollow shape to a StaticBody2D.

Warning: A non-uniformly scaled CollisionShape2D will likely not behave as expected. Make sure to keep its scale the same on all axes and adjust its shape resource instead.

Properties

BuildMode	build_mode	`0`
bool	disabled	`false`
bool	one_way_collision	`false`
float	one_way_collision_margin	`1.0`
PackedVector2Array	polygon	`PackedVector2Array()`

Enumerations

enum **BuildMode**: 🔗

BuildMode **BUILD_SOLIDS** = `0`

Collisions will include the polygon and its contained area. In this mode the node has the same effect as several ConvexPolygonShape2D nodes, one for each convex shape in the convex decomposition of the polygon (but without the overhead of multiple nodes).

BuildMode **BUILD_SEGMENTS** = `1`

Collisions will only include the polygon edges. In this mode the node has the same effect as a single ConcavePolygonShape2D made of segments, with the restriction that each segment (after the first one) starts where the previous one ends, and the last one ends where the first one starts (forming a closed but hollow polygon).

Property Descriptions

BuildMode **build_mode** = `0` 🔗

- void **set_build_mode**(value: BuildMode)
- BuildMode **get_build_mode**()

Collision build mode. Use one of the BuildMode constants.

---

bool **disabled** = `false` 🔗

- void **set_disabled**(value: bool)
- bool **is_disabled**()

If `true`, no collisions will be detected.

---

bool **one_way_collision** = `false` 🔗

- void **set_one_way_collision**(value: bool)
- bool **is_one_way_collision_enabled**()

If `true`, only edges that face up, relative to **CollisionPolygon2D**'s rotation, will collide with other objects.

Note: This property has no effect if this **CollisionPolygon2D** is a child of an Area2D node.

---

float **one_way_collision_margin** = `1.0` 🔗

- void **set_one_way_collision_margin**(value: float)
- float **get_one_way_collision_margin**()

The margin used for one-way collision (in pixels). Higher values will make the shape thicker, and work better for colliders that enter the polygon at a high velocity.

PackedVector2Array **polygon** = `PackedVector2Array()` 🔗

- void **set_polygon**(value: PackedVector2Array)
- PackedVector2Array **get_polygon**()

The polygon's list of vertices. Each point will be connected to the next, and the final point will be connected to the first.

Note: The returned vertices are in the local coordinate space of the given **CollisionPolygon2D**.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

CollisionPolygon3D

Inherits: Node3D **<** Node **<** Object

A node that provides a thickened polygon shape (a prism) to a CollisionObject3D parent.

Description

A node that provides a thickened polygon shape (a prism) to a CollisionObject3D parent and allows to edit it. The polygon can be concave or convex. This can give a detection shape to an Area3D or turn PhysicsBody3D into a solid object.

Warning: A non-uniformly scaled CollisionShape3D will likely not behave as expected. Make sure to keep its scale the same on all axes and adjust its shape resource instead.

Properties

float	depth	`1.0`
bool	disabled	`false`
float	margin	`0.04`
PackedVector2Array	polygon	`PackedVector2Array()`

Property Descriptions

float **depth** = `1.0`

- void **set_depth**(value: float)
- float **get_depth**()

Length that the resulting collision extends in either direction perpendicular to its 2D polygon.

---

bool **disabled** = `false`

- void **set_disabled**(value: bool)
- bool **is_disabled**()

If `true`, no collision will be produced.

---

float **margin** = `0.04`

- void **set_margin**(value: float)
- float **get_margin**()

The collision margin for the generated Shape3D. See Shape3D.margin for more details.

PackedVector2Array **polygon** = `PackedVector2Array()`

- void **set_polygon**(value: PackedVector2Array)
- PackedVector2Array **get_polygon**()

Array of vertices which define the 2D polygon in the local XY plane.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

CollisionShape2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A node that provides a Shape2D to a CollisionObject2D parent.

Description

A node that provides a Shape2D to a CollisionObject2D parent and allows to edit it. This can give a detection shape to an Area2D or turn a PhysicsBody2D into a solid object.

Tutorials

- Physics introduction
- 2D Dodge The Creeps Demo [https://godotengine.org/asset-library/asset/2712]
- 2D Pong Demo [https://godotengine.org/asset-library/asset/2728]
- 2D Kinematic Character Demo [https://godotengine.org/asset-library/asset/2719]

Properties

Color	debug_color	`Color(0, 0, 0, 1)`
bool	disabled	`false`
bool	one_way_collision	`false`

float	one_way_collision_margin	`1.0`
Shape2D	shape	

Property Descriptions

Color **debug_color** = `Color(0, 0, 0, 1)` 🔗

- void **set_debug_color**(value: Color)
- Color **get_debug_color**()

The collision shape debug color.

Note: The default value is ProjectSettings.debug/shapes/collision/shape_color. The `Color(0, 0, 0, 1)` value documented here is a placeholder, and not the actual default debug color.

---

bool **disabled** = `false` 🔗

- void **set_disabled**(value: bool)
- bool **is_disabled**()

A disabled collision shape has no effect in the world. This property should be changed with Object.set_deferred.

---

bool **one_way_collision** = `false` 🔗

- void **set_one_way_collision**(value: bool)
- bool **is_one_way_collision_enabled**()

Sets whether this collision shape should only detect collision on one side (top or bottom).

Note: This property has no effect if this **CollisionShape2D** is a child of an Area2D node.

---

float **one_way_collision_margin** = `1.0`

- void **set_one_way_collision_margin**(value: float)
- float **get_one_way_collision_margin**()

The margin used for one-way collision (in pixels). Higher values will make the shape thicker, and work better for colliders that enter the shape at a high velocity.

Shape2D **shape**

- void **set_shape**(value: Shape2D)
- Shape2D **get_shape**()

The actual shape owned by this collision shape.

CollisionShape3D

Inherits: Node3D **<** Node **<** Object

A node that provides a Shape3D to a CollisionObject3D parent.

Description

A node that provides a Shape3D to a CollisionObject3D parent and allows to edit it. This can give a detection shape to an Area3D or turn a PhysicsBody3D into a solid object.

Warning: A non-uniformly scaled **CollisionShape3D** will likely not behave as expected. Make sure to keep its scale the same on all axes and adjust its shape resource instead.

Tutorials

- Physics introduction
- 3D Kinematic Character Demo [https://godotengine.org/asset-library/asset/2739]
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

bool	disabled	`false`

Shape3D	shape

Methods

void	make_convex_from_siblings()
void	resource_changed(resource: Resource)

Property Descriptions

bool **disabled** = `false` 🔗

- void **set_disabled**(value: bool)
- bool **is_disabled**()

A disabled collision shape has no effect in the world.

Shape3D **shape** 🔗

- void **set_shape**(value: Shape3D)
- Shape3D **get_shape**()

The actual shape owned by this collision shape.

Method Descriptions

void **make_convex_from_siblings**() 🔗

Sets the collision shape's shape to the addition of all its convexed MeshInstance3D siblings geometry.

---

void **resource_changed**(resource: Resource) 🔗

Deprecated: Use Resource.changed instead.

This method does nothing.

ColorPicker

Inherits: VBoxContainer **<** BoxContainer **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

A widget that provides an interface for selecting or modifying a color.

Description

A widget that provides an interface for selecting or modifying a color. It can optionally provide functionalities like a color sampler (eyedropper), color modes, and presets.

Note: This control is the color picker widget itself. You can use a ColorPickerButton instead if you need a button that brings up a **ColorPicker** in a popup.

Tutorials

- Tween Interpolation Demo [https://godotengine.org/asset-library/asset/2733]

Properties

bool	can_add_swatches	`true`
Color	color	`Color(1, 1, 1, 1)`

ColorModeType	color_mode	`0`
bool	color_modes_visible	`true`
bool	deferred_mode	`false`
bool	edit_alpha	`true`
bool	hex_visible	`true`
PickerShapeType	picker_shape	`0`
bool	presets_visible	`true`
bool	sampler_visible	`true`
bool	sliders_visible	`true`

Methods

void	add_preset(color: Color)
void	add_recent_preset(color: Color)

void	erase_preset(color: Color)
void	erase_recent_preset(color: Color)
PackedColorArray	get_presets() const
PackedColorArray	get_recent_presets() const

Theme Properties

int	center_slider_grabbers	`1`
int	h_width	`30`
int	label_width	`10`
int	margin	`4`
int	sv_height	`256`
int	sv_width	`256`
Texture2D	add_preset	

Texture2D bar_arrow

Texture2D color_hue

Texture2D color_okhsl_hue

Texture2D expanded_arrow

Texture2D folded_arrow

Texture2D overbright_indicator

Texture2D picker_cursor

Texture2D sample_bg

Texture2D sample_revert

Texture2D screen_picker

Texture2D shape_circle

Texture2D shape_rect

Texture2D shape_rect_wheel

Signals

color_changed(color: Color) 🔗

Emitted when the color is changed.

---

preset_added(color: Color) 🔗

Emitted when a preset is added.

---

preset_removed(color: Color) 🔗

Emitted when a preset is removed.

Enumerations

enum **ColorModeType**: 🔗

ColorModeType **MODE_RGB** = `0`

Allows editing the color with Red/Green/Blue sliders.

ColorModeType **MODE_HSV** = `1`

Allows editing the color with Hue/Saturation/Value sliders.

ColorModeType **MODE_RAW** = `2`

Allows the color R, G, B component values to go beyond 1.0, which can be used for certain special operations that require it (like tinting without darkening or rendering sprites in HDR).

ColorModeType **MODE_OKHSL** = `3`

Allows editing the color with Hue/Saturation/Lightness sliders.

OKHSL is a new color space similar to HSL but that better match perception by leveraging the Oklab color space which is designed to be simple to use, while doing a good job at predicting perceived lightness, chroma and hue.

Okhsv and Okhsl color spaces
[https://bottosson.github.io/posts/colorpicker/]

---

enum **PickerShapeType**: 🔗

PickerShapeType **SHAPE_HSV_RECTANGLE** = `0`

HSV Color Model rectangle color space.

PickerShapeType **SHAPE_HSV_WHEEL** = `1`

HSV Color Model rectangle color space with a wheel.

PickerShapeType **SHAPE_VHS_CIRCLE** = `2`

HSV Color Model circle color space. Use Saturation as a radius.

PickerShapeType **SHAPE_OKHSL_CIRCLE** = `3`

HSL OK Color Model circle color space.

PickerShapeType **SHAPE_NONE** = `4`

The color space shape and the shape select button are hidden. Can't be selected from the shapes popup.

Property Descriptions

bool **can_add_swatches** = `true`

- void **set_can_add_swatches**(value: bool)
- bool **are_swatches_enabled**()

If `true`, it's possible to add presets under Swatches. If `false`, the button to add presets is disabled.

Color **color** = `Color(1, 1, 1, 1)`

- void **set_pick_color**(value: Color)
- Color **get_pick_color**()

The currently selected color.

ColorModeType **color_mode** = `0`

- void **set_color_mode**(value: ColorModeType)
- ColorModeType **get_color_mode**()

The currently selected color mode. See ColorModeType.

---

bool **color_modes_visible** = `true`

- void **set_modes_visible**(value: bool)
- bool **are_modes_visible**()

If `true`, the color mode buttons are visible.

---

bool **deferred_mode** = `false`

- void **set_deferred_mode**(value: bool)
- bool **is_deferred_mode**()

If `true`, the color will apply only after the user releases the mouse button, otherwise it will apply immediately even in mouse motion event (which can cause performance issues).

bool **edit_alpha** = `true`

- void **set_edit_alpha**(value: bool)
- bool **is_editing_alpha**()

If `true`, shows an alpha channel slider (opacity).

bool **hex_visible** = `true`

- void **set_hex_visible**(value: bool)
- bool **is_hex_visible**()

If `true`, the hex color code input field is visible.

PickerShapeType **picker_shape** = `0`

- void **set_picker_shape**(value: PickerShapeType)
- PickerShapeType **get_picker_shape**()

The shape of the color space view. See PickerShapeType.

bool **presets_visible** = `true`

- void **set_presets_visible**(value: bool)
- bool **are_presets_visible**()

If `true`, the Swatches and Recent Colors presets are visible.

---

bool **sampler_visible** = `true`

- void **set_sampler_visible**(value: bool)
- bool **is_sampler_visible**()

If `true`, the color sampler and color preview are visible.

---

bool **sliders_visible** = `true`

- void **set_sliders_visible**(value: bool)
- bool **are_sliders_visible**()

If `true`, the color sliders are visible.

Method Descriptions

void **add_preset**(color: Color)

Adds the given color to a list of color presets. The presets are displayed in the color picker and the user will be able to select them.

Note: The presets list is only for *this* color picker.

---

void **add_recent_preset**(color: Color)

Adds the given color to a list of color recent presets so that it can be picked later. Recent presets are the colors that were picked recently, a new preset is automatically created and added to recent presets when you pick a new color.

Note: The recent presets list is only for *this* color picker.

---

void **erase_preset**(color: Color)

Removes the given color from the list of color presets of this color picker.

---

void **erase_recent_preset**(color: Color)

Removes the given color from the list of color recent presets of this color picker.

PackedColorArray **get_presets**() const

Returns the list of colors in the presets of the color picker.

PackedColorArray **get_recent_presets**() const

Returns the list of colors in the recent presets of the color picker.

Theme Property Descriptions

int **center_slider_grabbers** = `1`

Overrides the Slider.center_grabber theme property of the sliders.

---

int **h_width** = `30`

The width of the hue selection slider.

---

int **label_width** = `10` 🔗

The minimum width of the color labels next to sliders.

---

int **margin** = `4` 🔗

The margin around the **ColorPicker**.

---

int **sv_height** = `256` 🔗

The height of the saturation-value selection box.

---

int **sv_width** = `256` 🔗

The width of the saturation-value selection box.

Texture2D **add_preset** 🔗

The icon for the "Add Preset" button.

Texture2D **bar_arrow** 🔗

The texture for the arrow grabber.

Texture2D **color_hue** 🔗

Custom texture for the hue selection slider on the right.

Texture2D **color_okhsl_hue** 🔗

Custom texture for the H slider in the OKHSL color mode.

Texture2D **expanded_arrow** 🔗

The icon for color preset drop down menu when expanded.

Texture2D **folded_arrow** 🔗

The icon for color preset drop down menu when folded.

Texture2D **overbright_indicator** 🔗

The indicator used to signalize that the color value is outside the 0-1 range.

Texture2D **picker_cursor** 🔗

The image displayed over the color box/circle (depending on the picker_shape), marking the currently selected color.

Texture2D **sample_bg** 🔗

Background panel for the color preview box (visible when the color is translucent).

Texture2D **sample_revert** 🔗

The icon for the revert button (visible on the middle of the "old" color when it differs from the currently selected color). This icon is modulated with a dark color if the "old" color is bright enough, so the icon should be bright to ensure visibility in both scenarios.

Texture2D **screen_picker** 🔗

The icon for the screen color picker button.

Texture2D **shape_circle** 🔗

The icon for circular picker shapes.

Texture2D **shape_rect** 🔗

The icon for rectangular picker shapes.

Texture2D **shape_rect_wheel** 🔗

The icon for rectangular wheel picker shapes.

ColorPickerButton

Inherits: Button **<** BaseButton **<** Control **<** CanvasItem **<** Node **<** Object

A button that brings up a ColorPicker when pressed.

Description

Encapsulates a ColorPicker, making it accessible by pressing a button. Pressing the button will toggle the ColorPicker's visibility.

See also BaseButton which contains common properties and methods associated with this node.

Note: By default, the button may not be wide enough for the color preview swatch to be visible. Make sure to set Control.custom_minimum_size to a big enough value to give the button enough space.

Tutorials

- 2D GD Paint Demo [https://godotengine.org/asset-library/asset/2768]
- GUI Drag And Drop Demo [https://godotengine.org/asset-library/asset/2767]

Properties

Color	color	`Color(0, 0, 0, 1)`

bool	edit_alpha	`true`
bool	toggle_mode	`true` (overrides BaseButton)

Methods

ColorPicker	get_picker()
PopupPanel	get_popup()

Theme Properties

Texture2D	bg

Signals

color_changed(color: Color) 🔗

Emitted when the color changes.

picker_created() 🔗

Emitted when the ColorPicker is created (the button is pressed for the first time).

popup_closed() 🔗

Emitted when the ColorPicker is closed.

Property Descriptions

Color **color** = `Color(0, 0, 0, 1)` 🔗

- void **set_pick_color**(value: Color)
- Color **get_pick_color**()

The currently selected color.

bool **edit_alpha** = `true` 🔗

- void **set_edit_alpha**(value: bool)
- bool **is_editing_alpha**()

If `true`, the alpha channel in the displayed ColorPicker will be visible.

Method Descriptions

ColorPicker **get_picker**() 🔗

Returns the ColorPicker that this node toggles.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their CanvasItem.visible property.

PopupPanel **get_popup**()

Returns the control's PopupPanel which allows you to connect to popup signals. This allows you to handle events when the ColorPicker is shown or hidden.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their Window.visible property.

Theme Property Descriptions

Texture2D **bg**

The background of the color preview rect on the button.

ColorRect

Inherits: Control **<** CanvasItem **<** Node **<** Object

A control that displays a solid color rectangle.

Description

Displays a rectangle filled with a solid color. If you need to display the border alone, consider using a Panel instead.

Tutorials

- 2D Dodge The Creeps Demo [https://godotengine.org/asset-library/asset/2712]

Properties

Color	color	`Color(1, 1, 1, 1)`

Property Descriptions

Color **color** = `Color(1, 1, 1, 1)`

- void **set_color**(value: Color)
- Color **get_color**()

The fill color of the rectangle.

ConeTwistJoint3D

Inherits: Joint3D **<** Node3D **<** Node **<** Object

A physics joint that connects two 3D physics bodies in a way that simulates a ball-and-socket joint.

Description

A physics joint that connects two 3D physics bodies in a way that simulates a ball-and-socket joint. The twist axis is initiated as the X axis of the **ConeTwistJoint3D**. Once the physics bodies swing, the twist axis is calculated as the middle of the X axes of the joint in the local space of the two physics bodies. Useful for limbs like shoulders and hips, lamps hanging off a ceiling, etc.

Properties

float	bias	`0.3`
float	relaxation	`1.0`
float	softness	`0.8`
float	swing_span	`0.785398`
float	twist_span	`3.14159`

Methods

float	get_param(param: Param) const
void	set_param(param: Param, value: float)

Enumerations

enum **Param**: 🔗

Param **PARAM_SWING_SPAN** = `0`

Swing is rotation from side to side, around the axis perpendicular to the twist axis.

The swing span defines, how much rotation will not get corrected along the swing axis.

Could be defined as looseness in the **ConeTwistJoint3D**.

If below 0.05, this behavior is locked.

Param **PARAM_TWIST_SPAN** = `1`

Twist is the rotation around the twist axis, this value defined how far the joint can twist.

Twist is locked if below 0.05.

Param **PARAM_BIAS** = `2`

The speed with which the swing or twist will take place.

The higher, the faster.

Param **PARAM_SOFTNESS** = `3`

The ease with which the joint starts to twist. If it's too low, it takes more force to start twisting the joint.

Param **PARAM_RELAXATION** = `4`

Defines, how fast the swing- and twist-speed-difference on both sides gets synced.

Param **PARAM_MAX** = `5`

Represents the size of the Param enum.

Property Descriptions

float **bias** = `0.3`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The speed with which the swing or twist will take place.

The higher, the faster.

---

float **relaxation** = `1.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

Defines, how fast the swing- and twist-speed-difference on both sides gets synced.

---

float **softness** = `0.8`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The ease with which the joint starts to twist. If it's too low, it takes more force to start twisting the joint.

---

float **swing_span** = `0.785398`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

Swing is rotation from side to side, around the axis perpendicular to the twist axis.

The swing span defines, how much rotation will not get corrected along the swing axis.

Could be defined as looseness in the **ConeTwistJoint3D**.

If below 0.05, this behavior is locked.

---

float **twist_span** = `3.14159`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

Twist is the rotation around the twist axis, this value defined how far the joint can twist.

Twist is locked if below 0.05.

Method Descriptions

float **get_param**(param: Param) const

Returns the value of the specified parameter.

void **set_param**(param: Param, value: float)

Sets the value of the specified parameter.

ConfirmationDialog

Inherits: AcceptDialog **<** Window **<** Viewport **<** Node **<** Object

Inherited By: EditorCommandPalette, EditorFileDialog, FileDialog, ScriptCreateDialog

A dialog used for confirmation of actions.

Description

A dialog used for confirmation of actions. This window is similar to AcceptDialog, but pressing its Cancel button can have a different outcome from pressing the OK button. The order of the two buttons varies depending on the host OS.

To get cancel action, you can use:

GDScript

```
get_cancel_button().pressed.connect(_on_canceled)
```

C#

```
GetCancelButton().Pressed += OnCanceled;
```

Properties

String	cancel_button_text	`"Cancel"`

Vector2i	min_size	`Vector2i(200, 70)` (overrides Window)
Vector2i	size	`Vector2i(200, 100)` (overrides Window)
String	title	`"Please Confirm..."` (overrides Window)

Methods

Button	get_cancel_button()

Property Descriptions

String **cancel_button_text** = `"Cancel"` 🔗

- void **set_cancel_button_text**(value: String)
- String **get_cancel_button_text**()

The text displayed by the cancel button (see get_cancel_button).

Method Descriptions

Button **get_cancel_button**() 🔗

Returns the cancel button.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their CanvasItem.visible property.

Container

Inherits: Control **<** CanvasItem **<** Node **<** Object

Inherited By: AspectRatioContainer, BoxContainer, CenterContainer, EditorProperty, FlowContainer, GraphElement, GridContainer, MarginContainer, PanelContainer, ScrollContainer, SplitContainer, SubViewportContainer, TabContainer

Base class for all GUI containers.

Description

Base class for all GUI containers. A **Container** automatically arranges its child controls in a certain way. This class can be inherited to make custom container types.

Tutorials

- Using Containers

Properties

MouseFilter	mouse_filter	`1` (overrides Control)

Methods

PackedInt32Array	_get_allowed_size_flags_horizontal() virtual const
PackedInt32Array	_get_allowed_size_flags_vertical() virtual const
void	fit_child_in_rect(child: Control, rect: Rect2)
void	queue_sort()

Signals

pre_sort_children()

Emitted when children are going to be sorted.

sort_children()

Emitted when sorting the children is needed.

Constants

NOTIFICATION_PRE_SORT_CHILDREN = `50`

Notification just before children are going to be sorted, in case there's something to process beforehand.

NOTIFICATION_SORT_CHILDREN = `51`

Notification for when sorting the children, it must be obeyed immediately.

Method Descriptions

PackedInt32Array **_get_allowed_size_flags_horizontal**() virtual const

Implement to return a list of allowed horizontal SizeFlags for child nodes. This doesn't technically prevent the usages of any other size flags, if your implementation requires that. This only limits the options available to the user in the Inspector dock.

Note: Having no size flags is equal to having Control.SIZE_SHRINK_BEGIN. As such, this value is always implicitly allowed.

PackedInt32Array **_get_allowed_size_flags_vertical**() virtual const

Implement to return a list of allowed vertical SizeFlags for child nodes. This doesn't technically prevent the usages of any other size flags, if your implementation requires that. This only limits the options available to the user in the Inspector dock.

Note: Having no size flags is equal to having Control.SIZE_SHRINK_BEGIN. As such, this value is always implicitly allowed.

void **fit_child_in_rect**(child: Control, rect: Rect2)

Fit a child control in a given rect. This is mainly a helper for creating custom container classes.

void **queue_sort**() 🔗

Queue resort of the contained children. This is called automatically anyway, but can be called upon request.

Control

Inherits: CanvasItem **<** Node **<** Object

Inherited By: BaseButton, ColorRect, Container, GraphEdit, ItemList, Label, LineEdit, MenuBar, NinePatchRect, Panel, Range, ReferenceRect, RichTextLabel, Separator, TabBar, TextEdit, TextureRect, Tree, VideoStreamPlayer

Base class for all GUI controls. Adapts its position and size based on its parent control.

Description

Base class for all UI-related nodes. **Control** features a bounding rectangle that defines its extents, an anchor position relative to its parent control or the current viewport, and offsets relative to the anchor. The offsets update automatically when the node, any of its parents, or the screen size change.

For more information on Godot's UI system, anchors, offsets, and containers, see the related tutorials in the manual. To build flexible UIs, you'll need a mix of UI elements that inherit from **Control** and Container nodes.

User Interface nodes and input

Godot propagates input events via viewports. Each Viewport is responsible for propagating InputEvents to their child nodes. As the SceneTree.root is a Window, this already happens automatically for all UI elements in your game.

Input events are propagated through the SceneTree from the root node to all child nodes by calling Node._input. For UI elements specifically, it makes more sense to override the virtual method _gui_input, which filters out unrelated input events, such as by checking z-order, mouse_filter, focus, or if the event was inside of the control's bounding box.

Call accept_event so no other node receives the event. Once you accept an input, it becomes handled so Node._unhandled_input will not process it.

Only one **Control** node can be in focus. Only the node in focus will receive events. To get the focus, call grab_focus. **Control** nodes lose focus when another node grabs it, or if you hide the node in focus.

Sets mouse_filter to MOUSE_FILTER_IGNORE to tell a **Control** node to ignore mouse or touch events. You'll need it if you place an icon on top of a button.

Theme resources change the Control's appearance. If you change the Theme on a **Control** node, it affects all of its children. To override some of the theme's parameters, call one of the `add_theme_*_override` methods, like add_theme_font_override. You can override the theme with the Inspector.

Note: Theme items are *not* Object properties. This means you can't access their values using Object.get and Object.set. Instead, use the `get_theme_*` and `add_theme_*_override` methods provided by this class.

Tutorials

- GUI documentation index
- Custom drawing in 2D
- Control node gallery
- Multiple resolutions
- All GUI Demos [https://github.com/godotengine/godot-demo-projects/tree/master/gui]

Properties

float	anchor_bottom	`0.0`
float	anchor_left	`0.0`
float	anchor_right	`0.0`
float	anchor_top	`0.0`
bool	auto_translate	
bool	clip_contents	`false`
Vector2	custom_minimum_size	`Vector2(0, 0)`
FocusMode	focus_mode	`0`
NodePath	focus_neighbor_bottom	`NodePath("")`
NodePath	focus_neighbor_left	`NodePath("")`
NodePath	focus_neighbor_right	`NodePath("")`
NodePath	focus_neighbor_top	`NodePath("")`
NodePath	focus_next	`NodePath("")`

NodePath	focus_previous	`NodePath("")`
Vector2	global_position	
GrowDirection	grow_horizontal	`1`
GrowDirection	grow_vertical	`1`
LayoutDirection	layout_direction	`0`
bool	localize_numeral_system	`true`
CursorShape	mouse_default_cursor_shape	`0`
MouseFilter	mouse_filter	`0`
bool	mouse_force_pass_scroll_events	`true`
float	offset_bottom	`0.0`
float	offset_left	`0.0`
float	offset_right	`0.0`
float	offset_top	`0.0`

PhysicsInterpolationMode	physics_interpolation_mode	`2` (overrides Node)
Vector2	pivot_offset	`Vector2(0, 0)`
Vector2	position	`Vector2(0, 0)`
float	rotation	`0.0`
float	rotation_degrees	
Vector2	scale	`Vector2(1, 1)`
Node	shortcut_context	
Vector2	size	`Vector2(0, 0)`
BitField[SizeFlags]	size_flags_horizontal	`1`
float	size_flags_stretch_ratio	`1.0`
BitField[SizeFlags]	size_flags_vertical	`1`
Theme	theme	

StringName	theme_type_variation	`&""`
String	tooltip_text	`""`

Methods

bool	_can_drop_data(at_position: Vector2, data: Variant) virtual const
void	_drop_data(at_position: Vector2, data: Variant) virtual
Variant	_get_drag_data(at_position: Vector2) virtual
Vector2	_get_minimum_size() virtual const
String	_get_tooltip(at_position: Vector2) virtual const
void	_gui_input(event: InputEvent) virtual
bool	_has_point(point: Vector2) virtual const
Object	_make_custom_tooltip(for_text: String) virtual const
Array[Vector3i]	_structured_text_parser(args: Array, text: String) virtual const
void	accept_event()

void	add_theme_color_override(name: StringName, color: Color)
void	add_theme_constant_override(name: StringName, constant: int)
void	add_theme_font_override(name: StringName, font: Font)
void	add_theme_font_size_override(name: StringName, font_size: int)
void	add_theme_icon_override(name: StringName, texture: Texture2D)
void	add_theme_stylebox_override(name: StringName, stylebox: StyleBox)
void	begin_bulk_theme_override()
void	end_bulk_theme_override()
Control	find_next_valid_focus() const
Control	find_prev_valid_focus() const
Control	find_valid_focus_neighbor(side: Side) const

void	force_drag(data: Variant, preview: Control)
float	get_anchor(side: Side) const
Vector2	get_begin() const
Vector2	get_combined_minimum_size() const
CursorShape	get_cursor_shape(position: Vector2 = Vector2(0, 0)) const
Vector2	get_end() const
NodePath	get_focus_neighbor(side: Side) const
Rect2	get_global_rect() const
Vector2	get_minimum_size() const
float	get_offset(offset: Side) const
Vector2	get_parent_area_size() const
Control	get_parent_control() const
Rect2	get_rect() const

Vector2	get_screen_position() const
Color	get_theme_color(name: StringName, theme_type: StringName = &"") const
int	get_theme_constant(name: StringName, theme_type: StringName = &"") const
float	get_theme_default_base_scale() const
Font	get_theme_default_font() const
int	get_theme_default_font_size() const
Font	get_theme_font(name: StringName, theme_type: StringName = &"") const
int	get_theme_font_size(name: StringName, theme_type: StringName = &"") const
Texture2D	get_theme_icon(name: StringName, theme_type: StringName = &"") const
StyleBox	get_theme_stylebox(name: StringName, theme_type: StringName = &"") const
String	get_tooltip(at_position: Vector2 = Vector2(0, 0)) const
void	grab_click_focus()

void	grab_focus()
bool	has_focus() const
bool	has_theme_color(name: StringName, theme_type: StringName = &"") const
bool	has_theme_color_override(name: StringName) const
bool	has_theme_constant(name: StringName, theme_type: StringName = &"") const
bool	has_theme_constant_override(name: StringName) const
bool	has_theme_font(name: StringName, theme_type: StringName = &"") const
bool	has_theme_font_override(name: StringName) const
bool	has_theme_font_size(name: StringName, theme_type: StringName = &"") const
bool	has_theme_font_size_override(name: StringName) const
bool	has_theme_icon(name: StringName, theme_type: StringName = &"") const

bool	has_theme_icon_override(name: StringName) const
bool	has_theme_stylebox(name: StringName, theme_type: StringName = &"") const
bool	has_theme_stylebox_override(name: StringName) const
bool	is_drag_successful() const
bool	is_layout_rtl() const
void	release_focus()
void	remove_theme_color_override(name: StringName)
void	remove_theme_constant_override(name: StringName)
void	remove_theme_font_override(name: StringName)
void	remove_theme_font_size_override(name: StringName)
void	remove_theme_icon_override(name: StringName)
void	remove_theme_stylebox_override(name: StringName)
void	reset_size()

void	set_anchor(side: Side, anchor: float, keep_offset: bool = false, push_opposite_anchor: bool = true)
void	set_anchor_and_offset(side: Side, anchor: float, offset: float, push_opposite_anchor: bool = false)
void	set_anchors_and_offsets_preset(preset: LayoutPreset, resize_mode: LayoutPresetMode = 0, margin: int = 0)
void	set_anchors_preset(preset: LayoutPreset, keep_offsets: bool = false)
void	set_begin(position: Vector2)
void	set_drag_forwarding(drag_func: Callable, can_drop_func: Callable, drop_func: Callable)
void	set_drag_preview(control: Control)
void	set_end(position: Vector2)
void	set_focus_neighbor(side: Side, neighbor: NodePath)
void	set_global_position(position: Vector2, keep_offsets: bool = false)
void	set_offset(side: Side, offset: float)

void	set_offsets_preset(preset: LayoutPreset, resize_mode: LayoutPresetMode = 0, margin: int = 0)
void	set_position(position: Vector2, keep_offsets: bool = false)
void	set_size(size: Vector2, keep_offsets: bool = false)
void	update_minimum_size()
void	warp_mouse(position: Vector2)

Signals

focus_entered() 🔗

Emitted when the node gains focus.

focus_exited() 🔗

Emitted when the node loses focus.

gui_input(event: InputEvent) 🔗

Emitted when the node receives an InputEvent.

minimum_size_changed() 🔗

Emitted when the node's minimum size changes.

mouse_entered() 🔗

Emitted when the mouse cursor enters the control's (or any child control's) visible area, that is not occluded behind other Controls or Windows, provided its mouse_filter lets the event reach it and regardless if it's currently focused or not.

Note: CanvasItem.z_index doesn't affect, which Control receives the signal.

---

mouse_exited() 🔗

Emitted when the mouse cursor leaves the control's (and all child control's) visible area, that is not occluded behind other Controls or Windows, provided its mouse_filter lets the event reach it and regardless if it's currently focused or not.

Note: CanvasItem.z_index doesn't affect, which Control receives the signal.

Note: If you want to check whether the mouse truly left the area, ignoring any top nodes, you can use code like this:

```
func _on_mouse_exited():
    if not Rect2(Vector2(),
size).has_point(get_local_mouse_position()):
        # Not hovering over area.
```

---

resized() 🔗

Emitted when the control changes size.

---

size_flags_changed() 🔗

Emitted when one of the size flags changes. See size_flags_horizontal and size_flags_vertical.

---

theme_changed() 🔗

Emitted when the NOTIFICATION_THEME_CHANGED notification is sent.

Enumerations

enum **FocusMode**: 🔗

FocusMode **FOCUS_NONE** = `0`

The node cannot grab focus. Use with focus_mode.

FocusMode **FOCUS_CLICK** = `1`

The node can only grab focus on mouse clicks. Use with focus_mode.

FocusMode **FOCUS_ALL** = `2`

The node can grab focus on mouse click, using the arrows and the Tab keys on the keyboard, or using the D-pad buttons on a gamepad. Use with focus_mode.

---

enum **CursorShape**: 🔗

CursorShape **CURSOR_ARROW** = `0`

Show the system's arrow mouse cursor when the user hovers the node. Use with mouse_default_cursor_shape.

CursorShape **CURSOR_IBEAM** = `1`

Show the system's I-beam mouse cursor when the user hovers the node. The I-beam pointer has a shape similar to "I". It tells the user they can highlight or insert text.

CursorShape **CURSOR_POINTING_HAND** = `2`

Show the system's pointing hand mouse cursor when the user hovers the node.

CursorShape **CURSOR_CROSS** = `3`

Show the system's cross mouse cursor when the user hovers the node.

CursorShape **CURSOR_WAIT** = `4`

Show the system's wait mouse cursor when the user hovers the node. Often an hourglass.

CursorShape **CURSOR_BUSY** = `5`

Show the system's busy mouse cursor when the user hovers the node. Often an arrow with a small hourglass.

CursorShape **CURSOR_DRAG** = `6`

Show the system's drag mouse cursor, often a closed fist or a cross symbol, when the user hovers the node. It tells the user they're currently dragging an item, like a node in the Scene dock.

CursorShape **CURSOR_CAN_DROP** = `7`

Show the system's drop mouse cursor when the user hovers the node. It can be an open hand. It tells the user they can drop an item they're currently grabbing, like a node in the Scene dock.

CursorShape **CURSOR_FORBIDDEN** = `8`

Show the system's forbidden mouse cursor when the user hovers the node. Often a crossed circle.

CursorShape **CURSOR_VSIZE** = `9`

Show the system's vertical resize mouse cursor when the user hovers the node. A double-headed vertical arrow. It tells the user they can resize the window or the panel vertically.

CursorShape **CURSOR_HSIZE** = `10`

Show the system's horizontal resize mouse cursor when the user hovers the node. A double-headed horizontal arrow. It tells the user they can resize the window or the panel horizontally.

CursorShape **CURSOR_BDIAGSIZE** = `11`

Show the system's window resize mouse cursor when the user hovers the node. The cursor is a double-headed arrow that goes from the bottom left to the top right. It tells the user they can resize the window or the panel both horizontally and vertically.

CursorShape **CURSOR_FDIAGSIZE** = `12`

Show the system's window resize mouse cursor when the user hovers the node. The cursor is a double-headed arrow that goes from the top left to the bottom right, the opposite of CURSOR_BDIAGSIZE. It tells the user they can resize the window or the panel both horizontally and vertically.

CursorShape **CURSOR_MOVE** = `13`

Show the system's move mouse cursor when the user hovers the node. It shows 2 double-headed arrows at a 90 degree angle. It tells the user they can move a UI element freely.

CursorShape **CURSOR_VSPLIT** = `14`

Show the system's vertical split mouse cursor when the user hovers the node. On Windows, it's the same as CURSOR_VSIZE.

CursorShape **CURSOR_HSPLIT** = `15`

Show the system's horizontal split mouse cursor when the user hovers the node. On Windows, it's the same as CURSOR_HSIZE.

CursorShape **CURSOR_HELP** = `16`

Show the system's help mouse cursor when the user hovers the node, a question mark.

---

enum **LayoutPreset**: 🔗

LayoutPreset **PRESET_TOP_LEFT** = `0`

Snap all 4 anchors to the top-left of the parent control's bounds. Use with set_anchors_preset.

LayoutPreset **PRESET_TOP_RIGHT** = `1`

Snap all 4 anchors to the top-right of the parent control's bounds. Use with set_anchors_preset.

LayoutPreset **PRESET_BOTTOM_LEFT** = `2`

Snap all 4 anchors to the bottom-left of the parent control's bounds. Use with set_anchors_preset.

LayoutPreset **PRESET_BOTTOM_RIGHT** = `3`

Snap all 4 anchors to the bottom-right of the parent control's bounds. Use with set_anchors_preset.

LayoutPreset **PRESET_CENTER_LEFT** = `4`

Snap all 4 anchors to the center of the left edge of the parent control's bounds. Use with set_anchors_preset.

LayoutPreset **PRESET_CENTER_TOP** = `5`

Snap all 4 anchors to the center of the top edge of the parent control's bounds. Use with set_anchors_preset.

LayoutPreset **PRESET_CENTER_RIGHT** = `6`

Snap all 4 anchors to the center of the right edge of the parent control's bounds. Use with set_anchors_preset.

LayoutPreset **PRESET_CENTER_BOTTOM** = `7`

Snap all 4 anchors to the center of the bottom edge of the parent control's bounds. Use with set_anchors_preset.

LayoutPreset **PRESET_CENTER** = `8`

Snap all 4 anchors to the center of the parent control's bounds. Use with set_anchors_preset.

LayoutPreset **PRESET_LEFT_WIDE** = `9`

Snap all 4 anchors to the left edge of the parent control. The left offset becomes relative to the left edge and the top offset relative to the top left corner of the node's parent. Use with set_anchors_preset.

LayoutPreset **PRESET_TOP_WIDE** = `10`

Snap all 4 anchors to the top edge of the parent control. The left offset becomes relative to the top left corner, the top offset relative to the top edge, and the right offset relative to the top right corner of the node's parent. Use with set_anchors_preset.

LayoutPreset **PRESET_RIGHT_WIDE** = `11`

Snap all 4 anchors to the right edge of the parent control. The right offset becomes relative to the right edge and the top offset relative to the top right corner of the node's parent. Use with set_anchors_preset.

LayoutPreset **PRESET_BOTTOM_WIDE** = `12`

Snap all 4 anchors to the bottom edge of the parent control. The left offset becomes relative to the bottom left corner, the bottom offset relative to the bottom edge, and the right offset relative to the bottom right corner of the node's parent. Use with set_anchors_preset.

LayoutPreset **PRESET_VCENTER_WIDE** = `13`

Snap all 4 anchors to a vertical line that cuts the parent control in half. Use with set_anchors_preset.

LayoutPreset **PRESET_HCENTER_WIDE** = `14`

Snap all 4 anchors to a horizontal line that cuts the parent control in half. Use with set_anchors_preset.

LayoutPreset **PRESET_FULL_RECT** = `15`

Snap all 4 anchors to the respective corners of the parent control. Set all 4 offsets to 0 after you applied this preset and the **Control** will fit its parent control. Use with set_anchors_preset.

enum **LayoutPresetMode**: 🔗

LayoutPresetMode **PRESET_MODE_MINSIZE** = `0`

The control will be resized to its minimum size.

LayoutPresetMode **PRESET_MODE_KEEP_WIDTH** = `1`

The control's width will not change.

LayoutPresetMode **PRESET_MODE_KEEP_HEIGHT** = `2`

The control's height will not change.

LayoutPresetMode **PRESET_MODE_KEEP_SIZE** = `3`

The control's size will not change.

flags **SizeFlags**: 🔗

SizeFlags **SIZE_SHRINK_BEGIN** = `0`

Tells the parent Container to align the node with its start, either the top or the left edge. It is mutually exclusive with SIZE_FILL and other shrink size flags, but can be used with SIZE_EXPAND in some containers. Use with size_flags_horizontal and size_flags_vertical.

Note: Setting this flag is equal to not having any size flags.

SizeFlags **SIZE_FILL** = `1`

Tells the parent Container to expand the bounds of this node to fill all the available space without pushing any other node. It is mutually exclusive with shrink size flags. Use with size_flags_horizontal and size_flags_vertical.

SizeFlags **SIZE_EXPAND** = `2`

Tells the parent Container to let this node take all the available space on the axis you flag. If multiple neighboring nodes are set to expand, they'll share the space based on their stretch ratio. See size_flags_stretch_ratio. Use with size_flags_horizontal and size_flags_vertical.

SizeFlags **SIZE_EXPAND_FILL** = `3`

Sets the node's size flags to both fill and expand. See SIZE_FILL and SIZE_EXPAND for more information.

SizeFlags **SIZE_SHRINK_CENTER** = `4`

Tells the parent Container to center the node in the available space. It is mutually exclusive with SIZE_FILL and other shrink size flags, but can be used with SIZE_EXPAND in some containers. Use with size_flags_horizontal and size_flags_vertical.

SizeFlags **SIZE_SHRINK_END** = `8`

Tells the parent Container to align the node with its end, either the bottom or the right edge. It is mutually exclusive with SIZE_FILL and other shrink size flags, but can be used with SIZE_EXPAND in some containers. Use with size_flags_horizontal and size_flags_vertical.

---

enum **MouseFilter**: 🔗

MouseFilter **MOUSE_FILTER_STOP** = `0`

The control will receive mouse movement input events and mouse button input events if clicked on through _gui_input. And the control will receive the mouse_entered and mouse_exited signals. These events are automatically marked as handled, and they will not propagate further to other controls. This also results in blocking signals in other controls.

MouseFilter **MOUSE_FILTER_PASS** = `1`

The control will receive mouse movement input events and mouse button input events if clicked on through _gui_input. And the control will receive the mouse_entered and mouse_exited signals. If this control does not handle the event, the parent control (if any) will be considered, and so on until there is no more parent control to potentially handle it. This also allows signals to fire in other controls. If no control handled it, the event will be passed to Node._shortcut_input for further processing.

MouseFilter **MOUSE_FILTER_IGNORE** = `2`

The control will not receive mouse movement input events and mouse button input events if clicked on through _gui_input. The control will also not receive the mouse_entered nor mouse_exited signals. This will not block other controls from receiving these events or firing the signals. Ignored events will not be handled automatically.

Note: If the control has received mouse_entered but not mouse_exited, changing the mouse_filter to MOUSE_FILTER_IGNORE will cause mouse_exited to be emitted.

---

enum **GrowDirection**: 🔗

GrowDirection **GROW_DIRECTION_BEGIN** = `0`

The control will grow to the left or top to make up if its minimum size is changed to be greater than its current size on the respective axis.

GrowDirection **GROW_DIRECTION_END** = `1`

The control will grow to the right or bottom to make up if its minimum size is changed to be greater than its current size on the respective axis.

GrowDirection **GROW_DIRECTION_BOTH** = `2`

The control will grow in both directions equally to make up if its minimum size is changed to be greater than its current size.

---

enum **Anchor**: 🔗

Anchor **ANCHOR_BEGIN** = `0`

Snaps one of the 4 anchor's sides to the origin of the node's `Rect`, in the top left. Use it with one of the `anchor_*` member variables, like anchor_left. To change all 4 anchors at once, use set_anchors_preset.

Anchor **ANCHOR_END** = `1`

Snaps one of the 4 anchor's sides to the end of the node's `Rect`, in the bottom right. Use it with one of the `anchor_*` member variables, like anchor_left. To change all 4 anchors at once, use set_anchors_preset.

---

enum **LayoutDirection**: 🔗

LayoutDirection **LAYOUT_DIRECTION_INHERITED** = `0`

Automatic layout direction, determined from the parent control layout direction.

LayoutDirection **LAYOUT_DIRECTION_LOCALE** = `1`

Automatic layout direction, determined from the current locale.

LayoutDirection **LAYOUT_DIRECTION_LTR** = `2`

Left-to-right layout direction.

LayoutDirection **LAYOUT_DIRECTION_RTL** = `3`

Right-to-left layout direction.

---

enum **TextDirection**: 🔗

TextDirection **TEXT_DIRECTION_INHERITED** = `3`

Text writing direction is the same as layout direction.

TextDirection **TEXT_DIRECTION_AUTO** = `0`

Automatic text writing direction, determined from the current locale and text content.

TextDirection **TEXT_DIRECTION_LTR** = `1`

Left-to-right text writing direction.

TextDirection **TEXT_DIRECTION_RTL** = `2`

Right-to-left text writing direction.

Constants

NOTIFICATION_RESIZED = `40` 🔗

Sent when the node changes size. Use size to get the new size.

NOTIFICATION_MOUSE_ENTER = `41` 🔗

Sent when the mouse cursor enters the control's (or any child control's) visible area, that is not occluded behind other Controls or Windows, provided its mouse_filter lets the event reach it and regardless if it's currently focused or not.

Note: CanvasItem.z_index doesn't affect which Control receives the notification.

See also NOTIFICATION_MOUSE_ENTER_SELF.

NOTIFICATION_MOUSE_EXIT = `42` 🔗

Sent when the mouse cursor leaves the control's (and all child control's) visible area, that is not occluded behind other Controls or Windows, provided its mouse_filter lets the event reach it and regardless if it's currently focused or not.

Note: CanvasItem.z_index doesn't affect which Control receives the notification.

See also NOTIFICATION_MOUSE_EXIT_SELF.

NOTIFICATION_MOUSE_ENTER_SELF = `60` 🔗

Experimental: The reason this notification is sent may change in the future.

Sent when the mouse cursor enters the control's visible area, that is not occluded behind other Controls or Windows, provided its mouse_filter lets the event reach it and regardless if it's currently focused or not.

Note: CanvasItem.z_index doesn't affect which Control receives the notification.

See also NOTIFICATION_MOUSE_ENTER.

NOTIFICATION_MOUSE_EXIT_SELF = `61` 🔗

Experimental: The reason this notification is sent may change in the future.

Sent when the mouse cursor leaves the control's visible area, that is not occluded behind other Controls or Windows, provided its mouse_filter lets the event reach it and regardless if it's currently focused or not.

Note: CanvasItem.z_index doesn't affect which Control receives the notification.

See also NOTIFICATION_MOUSE_EXIT.

NOTIFICATION_FOCUS_ENTER = `43` 🔗

Sent when the node grabs focus.

NOTIFICATION_FOCUS_EXIT = `44` 🔗

Sent when the node loses focus.

NOTIFICATION_THEME_CHANGED = `45` 🔗

Sent when the node needs to refresh its theme items. This happens in one of the following cases:

- The theme property is changed on this node or any of its ancestors.
- The theme_type_variation property is changed on this node.
- One of the node's theme property overrides is changed.
- The node enters the scene tree.

Note: As an optimization, this notification won't be sent from changes that occur while this node is outside of the scene tree. Instead, all of the theme item updates can be applied at once when the node enters the scene tree.

Note: This notification is received alongside Node.NOTIFICATION_ENTER_TREE, so if you are instantiating a scene, the child nodes will not be initialized yet. You can use it to setup theming for this node, child nodes created from script, or if you want to access child nodes added in the editor, make sure the node is ready using Node.is_node_ready.

```
func _notification(what):
    if what == NOTIFICATION_THEME_CHANGED:
        if not is_node_ready():
            await ready # Wait until ready signal.
        $Label.add_theme_color_override("font_color", Color.YELLOW)
```

NOTIFICATION_SCROLL_BEGIN = `47` 🔗

Sent when this node is inside a ScrollContainer which has begun being scrolled when dragging the scrollable area *with a touch event*. This notification is *not* sent when scrolling by dragging the scrollbar, scrolling with the mouse wheel or scrolling with keyboard/gamepad events.

Note: This signal is only emitted on Android or iOS, or on desktop/web platforms when ProjectSettings.input_devices/pointing/emulate_touch_from_mouse is enabled.

NOTIFICATION_SCROLL_END = `48`

Sent when this node is inside a ScrollContainer which has stopped being scrolled when dragging the scrollable area *with a touch event*. This notification is *not* sent when scrolling by dragging the scrollbar, scrolling with the mouse wheel or scrolling with keyboard/gamepad events.

Note: This signal is only emitted on Android or iOS, or on desktop/web platforms when ProjectSettings.input_devices/pointing/emulate_touch_from_mouse is enabled.

NOTIFICATION_LAYOUT_DIRECTION_CHANGED = `49`

Sent when control layout direction is changed.

Property Descriptions

float **anchor_bottom** = `0.0`

- float **get_anchor**(side: Side) const

Anchors the bottom edge of the node to the origin, the center, or the end of its parent control. It changes how the bottom offset updates when the node moves or changes size. You can use one of the Anchor constants for convenience.

---

float **anchor_left** = `0.0`

- float **get_anchor**(side: Side) const

Anchors the left edge of the node to the origin, the center or the end of its parent control. It changes how the left offset updates when the node moves or changes size. You can use one of the Anchor constants for convenience.

---

float **anchor_right** = `0.0`

- float **get_anchor**(side: Side) const

Anchors the right edge of the node to the origin, the center or the end of its parent control. It changes how the right offset updates when the node moves or changes size. You can use one of the Anchor constants for convenience.

---

float **anchor_top** = `0.0`

- float **get_anchor**(side: Side) const

Anchors the top edge of the node to the origin, the center or the end of its parent control. It changes how the top offset updates when the node moves or changes size. You can use one of the Anchor constants for convenience.

---

bool **auto_translate**

- void **set_auto_translate**(value: bool)
- bool **is_auto_translating**()

Deprecated: Use Node.auto_translate_mode instead.

Toggles if any text should automatically change to its translated version depending on the current locale.

---

bool **clip_contents** = `false`

- void **set_clip_contents**(value: bool)
- bool **is_clipping_contents**()

Enables whether rendering of CanvasItem based children should be clipped to this control's rectangle. If `true`, parts of a child which would be visibly outside of this control's rectangle will not be rendered and won't receive input.

Vector2 **custom_minimum_size** = `Vector2(0, 0)`

- void **set_custom_minimum_size**(value: Vector2)
- Vector2 **get_custom_minimum_size**()

The minimum size of the node's bounding rectangle. If you set it to a value greater than `(0, 0)`, the node's bounding rectangle will always have at least this size. Note that **Control** nodes have their internal minimum size returned by get_minimum_size. It depends on the control's contents, like text, textures, or style boxes. The actual minimum size is the maximum value of this property and the internal minimum size (see get_combined_minimum_size).

FocusMode **focus_mode** = `0`

- void **set_focus_mode**(value: FocusMode)
- FocusMode **get_focus_mode**()

The focus access mode for the control (None, Click or All). Only one Control can be focused at the same time, and it will receive keyboard, gamepad, and mouse signals.

NodePath **focus_neighbor_bottom** = `NodePath("")`

- void **set_focus_neighbor**(side: Side, neighbor: NodePath)
- NodePath **get_focus_neighbor**(side: Side) const

Tells Godot which node it should give focus to if the user presses the down arrow on the keyboard or down on a gamepad by default. You can change the key by editing the ProjectSettings.input/ui_down input action. The node must be a **Control**. If this property is not set, Godot will give focus to the closest **Control** to the bottom of this one.

NodePath **focus_neighbor_left** = `NodePath("")`

- void **set_focus_neighbor**(side: Side, neighbor: NodePath)

- NodePath **get_focus_neighbor**(side: Side) const

Tells Godot which node it should give focus to if the user presses the left arrow on the keyboard or left on a gamepad by default. You can change the key by editing the ProjectSettings.input/ui_left input action. The node must be a **Control**. If this property is not set, Godot will give focus to the closest **Control** to the left of this one.

NodePath **focus_neighbor_right** = `NodePath("")`

- void **set_focus_neighbor**(side: Side, neighbor: NodePath)
- NodePath **get_focus_neighbor**(side: Side) const

Tells Godot which node it should give focus to if the user presses the right arrow on the keyboard or right on a gamepad by default. You can change the key by editing the ProjectSettings.input/ui_right input action. The node must be a **Control**. If this property is not set, Godot will give focus to the closest **Control** to the right of this one.

NodePath **focus_neighbor_top** = `NodePath("")`

- void **set_focus_neighbor**(side: Side, neighbor: NodePath)
- NodePath **get_focus_neighbor**(side: Side) const

Tells Godot which node it should give focus to if the user presses the top arrow on the keyboard or top on a gamepad by default. You can change the key by editing the ProjectSettings.input/ui_up input action. The node must be a **Control**. If this property is not set, Godot will give focus to the closest **Control** to the top of this one.

NodePath **focus_next** = `NodePath("")`

- void **set_focus_next**(value: NodePath)
- NodePath **get_focus_next**()

Tells Godot which node it should give focus to if the user presses `Tab` on a keyboard by default. You can change the key by editing the

ProjectSettings.input/ui_focus_next input action.

If this property is not set, Godot will select a "best guess" based on surrounding nodes in the scene tree.

NodePath **focus_previous** = `NodePath("")`

- void **set_focus_previous**(value: NodePath)
- NodePath **get_focus_previous**()

Tells Godot which node it should give focus to if the user presses `Shift + Tab` on a keyboard by default. You can change the key by editing the ProjectSettings.input/ui_focus_prev input action.

If this property is not set, Godot will select a "best guess" based on surrounding nodes in the scene tree.

Vector2 **global_position**

- Vector2 **get_global_position**()

The node's global position, relative to the world (usually to the CanvasLayer).

GrowDirection **grow_horizontal** = `1`

- void **set_h_grow_direction**(value: GrowDirection)
- GrowDirection **get_h_grow_direction**()

Controls the direction on the horizontal axis in which the control should grow if its horizontal minimum size is changed to be greater than its current size, as the control always has to be at least the minimum size.

GrowDirection **grow_vertical** = `1`

- void **set_v_grow_direction**(value: GrowDirection)

- GrowDirection **get_v_grow_direction**()

Controls the direction on the vertical axis in which the control should grow if its vertical minimum size is changed to be greater than its current size, as the control always has to be at least the minimum size.

LayoutDirection **layout_direction** = `0` 🔗

- void **set_layout_direction**(value: LayoutDirection)
- LayoutDirection **get_layout_direction**()

Controls layout direction and text writing direction. Right-to-left layouts are necessary for certain languages (e.g. Arabic and Hebrew).

---

bool **localize_numeral_system** = `true` 🔗

- void **set_localize_numeral_system**(value: bool)
- bool **is_localizing_numeral_system**()

If `true`, automatically converts code line numbers, list indices, SpinBox and ProgressBar values from the Western Arabic (0..9) to the numeral systems used in current locale.

Note: Numbers within the text are not automatically converted, it can be done manually, using TextServer.format_number.

CursorShape **mouse_default_cursor_shape** = `0` 🔗

- void **set_default_cursor_shape**(value: CursorShape)
- CursorShape **get_default_cursor_shape**()

The default cursor shape for this control. Useful for Godot plugins and applications or games that use the system's mouse cursors.

Note: On Linux, shapes may vary depending on the cursor theme of the system.

MouseFilter **mouse_filter** = `0` 🔗

- void **set_mouse_filter**(value: MouseFilter)
- MouseFilter **get_mouse_filter**()

Controls whether the control will be able to receive mouse button input events through _gui_input and how these events should be handled. Also controls whether the control can receive the mouse_entered, and mouse_exited signals. See the constants to learn what each does.

---

bool **mouse_force_pass_scroll_events** = `true` 🔗

- void **set_force_pass_scroll_events**(value: bool)
- bool **is_force_pass_scroll_events**()

When enabled, scroll wheel events processed by _gui_input will be passed to the parent control even if mouse_filter is set to MOUSE_FILTER_STOP. As it defaults to true, this allows nested scrollable containers to work out of the box.

You should disable it on the root of your UI if you do not want scroll events to go to the Node._unhandled_input processing.

---

float **offset_bottom** = `0.0` 🔗

- void **set_offset**(side: Side, offset: float)
- float **get_offset**(offset: Side) const

Distance between the node's bottom edge and its parent control, based on anchor_bottom.

Offsets are often controlled by one or multiple parent Container nodes, so you should not modify them manually if your node is a direct child of a Container. Offsets update automatically when you move or resize the node.

---

float **offset_left** = `0.0` 🔗

- void **set_offset**(side: Side, offset: float)
- float **get_offset**(offset: Side) const

Distance between the node's left edge and its parent control, based on anchor_left.

Offsets are often controlled by one or multiple parent Container nodes, so you should not modify them manually if your node is a direct child of a Container. Offsets update automatically when you move or resize the node.

float **offset_right** = `0.0` 🔗

- void **set_offset**(side: Side, offset: float)
- float **get_offset**(offset: Side) const

Distance between the node's right edge and its parent control, based on anchor_right.

Offsets are often controlled by one or multiple parent Container nodes, so you should not modify them manually if your node is a direct child of a Container. Offsets update automatically when you move or resize the node.

float **offset_top** = `0.0` 🔗

- void **set_offset**(side: Side, offset: float)
- float **get_offset**(offset: Side) const

Distance between the node's top edge and its parent control, based on anchor_top.

Offsets are often controlled by one or multiple parent Container nodes, so you should not modify them manually if your node is a direct child of a Container. Offsets update automatically when you move or resize the node.

Vector2 **pivot_offset** = `Vector2(0, 0)` 🔗

- void **set_pivot_offset**(value: Vector2)
- Vector2 **get_pivot_offset**()

By default, the node's pivot is its top-left corner. When you change its rotation or scale, it will rotate or scale around this pivot. Set this property to size / 2 to pivot around the Control's center.

Vector2 **position** = `Vector2(0, 0)` 🔗

- Vector2 **get_position**()

The node's position, relative to its containing node. It corresponds to the rectangle's top-left corner. The property is not affected by pivot_offset.

---

float **rotation** = `0.0` 🔗

- void **set_rotation**(value: float)
- float **get_rotation**()

The node's rotation around its pivot, in radians. See pivot_offset to change the pivot's position.

Note: This property is edited in the inspector in degrees. If you want to use degrees in a script, use rotation_degrees.

---

float **rotation_degrees** 🔗

- void **set_rotation_degrees**(value: float)
- float **get_rotation_degrees**()

Helper property to access rotation in degrees instead of radians.

Vector2 **scale** = `Vector2(1, 1)` 🔗

- void **set_scale**(value: Vector2)
- Vector2 **get_scale**()

The node's scale, relative to its size. Change this property to scale the node around its pivot_offset. The Control's tooltip_text will also scale according to this value.

Note: This property is mainly intended to be used for animation purposes. To support multiple resolutions in your project, use an appropriate viewport stretch mode as described in the documentation instead of scaling Controls individually.

Note: FontFile.oversampling does *not* take **Control** scale into account. This means that scaling up/down will cause bitmap fonts and rasterized (non-MSDF) dynamic fonts to appear blurry or pixelated. To ensure text remains crisp regardless of scale, you can enable MSDF font rendering by enabling ProjectSettings.gui/theme/default_font_multichannel_signed_distance_field (applies to the default project font only), or enabling **Multichannel Signed Distance Field** in the import options of a DynamicFont for custom fonts. On system fonts, SystemFont.multichannel_signed_distance_field can be enabled in the inspector.

Note: If the Control node is a child of a Container node, the scale will be reset to `Vector2(1, 1)` when the scene is instantiated. To set the Control's scale when it's instantiated, wait for one frame using `await get_tree().process_frame` then set its scale property.

Node **shortcut_context** 🔗

- void **set_shortcut_context**(value: Node)
- Node **get_shortcut_context**()

The Node which must be a parent of the focused **Control** for the shortcut to be activated. If `null`, the shortcut can be activated when any control is focused (a global shortcut). This allows shortcuts to be accepted only when the user has a certain area of the GUI focused.

Vector2 **size** = `Vector2(0, 0)`

- Vector2 **get_size**()

The size of the node's bounding rectangle, in the node's coordinate system. Container nodes update this property automatically.

BitField[SizeFlags] **size_flags_horizontal** = `1`

- void **set_h_size_flags**(value: BitField[SizeFlags])
- BitField[SizeFlags] **get_h_size_flags**()

Tells the parent Container nodes how they should resize and place the node on the X axis. Use a combination of the SizeFlags constants to change the flags. See the constants to learn what each does.

---

float **size_flags_stretch_ratio** = `1.0`

- void **set_stretch_ratio**(value: float)
- float **get_stretch_ratio**()

If the node and at least one of its neighbors uses the SIZE_EXPAND size flag, the parent Container will let it take more or less space depending on this property. If this node has a stretch ratio of 2 and its neighbor a ratio of 1, this node will take two thirds of the available space.

BitField[SizeFlags] **size_flags_vertical** = `1`

- void **set_v_size_flags**(value: BitField[SizeFlags])
- BitField[SizeFlags] **get_v_size_flags**()

Tells the parent Container nodes how they should resize and place the node on the Y axis. Use a combination of the SizeFlags constants to change the flags. See the constants to learn what each does.

Theme **theme** 🔗

- void **set_theme**(value: Theme)
- Theme **get_theme**()

The Theme resource this node and all its **Control** and Window children use. If a child node has its own Theme resource set, theme items are merged with child's definitions having higher priority.

Note: Window styles will have no effect unless the window is embedded.

StringName **theme_type_variation** = `&""` 🔗

- void **set_theme_type_variation**(value: StringName)
- StringName **get_theme_type_variation**()

The name of a theme type variation used by this **Control** to look up its own theme items. When empty, the class name of the node is used (e.g. `Button` for the Button control), as well as the class names of all parent classes (in order of inheritance).

When set, this property gives the highest priority to the type of the specified name. This type can in turn extend another type, forming a dependency chain. See Theme.set_type_variation. If the theme item cannot be found using this type or its base types, lookup falls back on the class names.

Note: To look up **Control**'s own items use various `get_theme_*` methods without specifying `theme_type`.

Note: Theme items are looked for in the tree order, from branch to root, where each **Control** node is checked for its theme property. The earliest match against any type/class name is returned. The project-level Theme and the default Theme are checked last.

String **tooltip_text** = `""` 🔗

- void **set_tooltip_text**(value: String)

- String **get_tooltip_text**()

The default tooltip text. The tooltip appears when the user's mouse cursor stays idle over this control for a few moments, provided that the mouse_filter property is not MOUSE_FILTER_IGNORE. The time required for the tooltip to appear can be changed with the ProjectSettings.gui/timers/tooltip_delay_sec option. See also get_tooltip.

The tooltip popup will use either a default implementation, or a custom one that you can provide by overriding _make_custom_tooltip. The default tooltip includes a PopupPanel and Label whose theme properties can be customized using Theme methods with the `"TooltipPanel"` and `"TooltipLabel"` respectively. For example:

GDScript

```
var style_box = StyleBoxFlat.new()
style_box.set_bg_color(Color(1, 1, 0))
style_box.set_border_width_all(2)
# We assume here that the `theme` property has been assigned a
custom Theme beforehand.
theme.set_stylebox("panel", "TooltipPanel", style_box)
theme.set_color("font_color", "TooltipLabel", Color(0, 1, 1))
```

C#

```
var styleBox = new StyleBoxFlat();
styleBox.SetBgColor(new Color(1, 1, 0));
styleBox.SetBorderWidthAll(2);
// We assume here that the `Theme` property has been assigned a
custom Theme beforehand.
Theme.SetStyleBox("panel", "TooltipPanel", styleBox);
Theme.SetColor("font_color", "TooltipLabel", new Color(0, 1, 1));
```

Method Descriptions

bool **_can_drop_data**(at_position: Vector2, data: Variant) virtual const 🔗

Godot calls this method to test if `data` from a control's _get_drag_data can be dropped at `at_position`. `at_position` is local to this control.

This method should only be used to test the data. Process the data in _drop_data.

GDScript

```
func _can_drop_data(position, data):
    # Check position if it is relevant to you
    # Otherwise, just check data
    return typeof(data) == TYPE_DICTIONARY and data.has("expected")
```

C#

```
public override bool _CanDropData(Vector2 atPosition, Variant data)
{
    // Check position if it is relevant to you
    // Otherwise, just check data
    return data.VariantType == Variant.Type.Dictionary &&
data.AsGodotDictionary().ContainsKey("expected");
}
```

void **_drop_data**(at_position: Vector2, data: Variant) virtual 🔗

Godot calls this method to pass you the `data` from a control's _get_drag_data result. Godot first calls _can_drop_data to test if `data` is allowed to drop at `at_position` where `at_position` is local to this control.

GDScript

```
func _can_drop_data(position, data):
    return typeof(data) == TYPE_DICTIONARY and data.has("color")

func _drop_data(position, data):
    var color = data["color"]
```

C#

```
public override bool _CanDropData(Vector2 atPosition, Variant data)
{
    return data.VariantType == Variant.Type.Dictionary &&
dict.AsGodotDictionary().ContainsKey("color");
}

public override void _DropData(Vector2 atPosition, Variant data)
{
    Color color = data.AsGodotDictionary()["color"].AsColor();
}
```

Variant **_get_drag_data**(at_position: Vector2) virtual 🔗

Godot calls this method to get data that can be dragged and dropped onto controls that expect drop data. Returns `null` if there is no data to drag. Controls that want to receive drop data should implement _can_drop_data and _drop_data. `at_position` is local to this control. Drag may be forced with force_drag.

A preview that will follow the mouse that should represent the data can be set with set_drag_preview. A good time to set the preview is in this method.

GDScript

```
func _get_drag_data(position):
    var mydata = make_data() # This is your custom method generating
the drag data.
    set_drag_preview(make_preview(mydata)) # This is your custom
method generating the preview of the drag data.
    return mydata
```

C#

```
public override Variant _GetDragData(Vector2 atPosition)
{
    var myData = MakeData(); // This is your custom method
generating the drag data.
    SetDragPreview(MakePreview(myData)); // This is your custom
method generating the preview of the drag data.
    return myData;
}
```

Vector2 **_get_minimum_size**() virtual const 🔗

Virtual method to be implemented by the user. Returns the minimum size for this control. Alternative to custom_minimum_size for controlling minimum size via code. The actual minimum size will be the max value of these two (in each axis separately).

If not overridden, defaults to Vector2.ZERO.

Note: This method will not be called when the script is attached to a **Control** node that already overrides its minimum size (e.g. Label, Button, PanelContainer etc.). It can only be used with most basic GUI nodes, like **Control**, Container, Panel etc.

String **_get_tooltip**(at_position: Vector2) virtual const

Virtual method to be implemented by the user. Returns the tooltip text for the position `at_position` in control's local coordinates, which will typically appear when the cursor is resting over this control. See get_tooltip.

Note: If this method returns an empty String, no tooltip is displayed.

---

void **_gui_input**(event: InputEvent) virtual

Virtual method to be implemented by the user. Use this method to process and accept inputs on UI elements. See accept_event.

Example usage for clicking a control:

GDScript

```
func _gui_input(event):
    if event is InputEventMouseButton:
        if event.button_index == MOUSE_BUTTON_LEFT and
event.pressed:
            print("I've been clicked D:")
```

C#

```
public override void _GuiInput(InputEvent @event)
{
    if (@event is InputEventMouseButton mb)
    {
        if (mb.ButtonIndex == MouseButton.Left && mb.Pressed)
        {
            GD.Print("I've been clicked D:");
        }
    }
}
```

The event won't trigger if:

* clicking outside the control (see _has_point);

* control has mouse_filter set to MOUSE_FILTER_IGNORE;

* control is obstructed by another **Control** on top of it, which doesn't have mouse_filter set to MOUSE_FILTER_IGNORE;

* control's parent has mouse_filter set to MOUSE_FILTER_STOP or has accepted the event;

* it happens outside the parent's rectangle and the parent has either clip_contents enabled.

Note: Event position is relative to the control origin.

---

bool **_has_point**(point: Vector2) virtual const

Virtual method to be implemented by the user. Returns whether the given `point` is inside this control.

If not overridden, default behavior is checking if the point is within control's Rect.

Note: If you want to check if a point is inside the control, you can use `Rect2(Vector2.ZERO, size).has_point(point)`.

Object **_make_custom_tooltip**(for_text: String) virtual const

Virtual method to be implemented by the user. Returns a **Control** node that should be used as a tooltip instead of the default one. The `for_text` includes the contents of the tooltip_text property.

The returned node must be of type **Control** or Control-derived. It can have child nodes of any type. It is freed when the tooltip disappears, so make sure you always provide a new instance (if you want to use a pre-existing node from your scene tree, you can duplicate it and pass the duplicated instance). When `null` or a non-Control node is returned, the default tooltip will be used instead.

The returned node will be added as child to a PopupPanel, so you should only provide the contents of that panel. That PopupPanel can be themed using Theme.set_stylebox for the type `"TooltipPanel"` (see tooltip_text for an example).

Note: The tooltip is shrunk to minimal size. If you want to ensure it's fully visible, you might want to set its custom_minimum_size to some non-zero value.

Note: The node (and any relevant children) should be CanvasItem.visible when returned, otherwise, the viewport that instantiates it will not be able to calculate its minimum size reliably.

Example of usage with a custom-constructed node:

GDScript

```
func _make_custom_tooltip(for_text):
    var label = Label.new()
    label.text = for_text
    return label
```

C#

```
public override Control _MakeCustomTooltip(string forText)
{
    var label = new Label();
    label.Text = forText;
    return label;
}
```

Example of usage with a custom scene instance:

GDScript

```
func _make_custom_tooltip(for_text):
    var tooltip =
preload("res://some_tooltip_scene.tscn").instantiate()
    tooltip.get_node("Label").text = for_text
    return tooltip
```

C#

```
public override Control _MakeCustomTooltip(string forText)
{
    Node tooltip = ResourceLoader.Load<PackedScene>
("res://some_tooltip_scene.tscn").Instantiate();
    tooltip.GetNode<Label>("Label").Text = forText;
    return tooltip;
}
```

Array[Vector3i] **_structured_text_parser**(args: Array, text: String) virtual const

User defined BiDi algorithm override function.

Returns an Array of Vector3i text ranges and text base directions, in the left-to-right order. Ranges should cover full source `text` without overlaps. BiDi algorithm will be used on each range separately.

void **accept_event**()

Marks an input event as handled. Once you accept an input event, it stops propagating, even to nodes listening to Node._unhandled_input or Node._unhandled_key_input.

Note: This does not affect the methods in Input, only the way events are propagated.

void **add_theme_color_override**(name: StringName, color: Color)

Creates a local override for a theme Color with the specified `name`. Local overrides always take precedence when fetching theme items for the control. An override can be removed with remove_theme_color_override.

See also get_theme_color.

Example of overriding a label's color and resetting it later:

GDScript

```
# Given the child Label node "MyLabel", override its font color with
a custom value.
$MyLabel.add_theme_color_override("font_color", Color(1, 0.5, 0))
# Reset the font color of the child label.
$MyLabel.remove_theme_color_override("font_color")
# Alternatively it can be overridden with the default value from the
Label type.
$MyLabel.add_theme_color_override("font_color",
get_theme_color("font_color", "Label"))
```

C#

```
// Given the child Label node "MyLabel", override its font color
with a custom value.
GetNode<Label>("MyLabel").AddThemeColorOverride("font_color", new
Color(1, 0.5f, 0));
// Reset the font color of the child label.
GetNode<Label>("MyLabel").RemoveThemeColorOverride("font_color");
// Alternatively it can be overridden with the default value from
the Label type.
GetNode<Label>("MyLabel").AddThemeColorOverride("font_color",
GetThemeColor("font_color", "Label"));
```

---

void **add_theme_constant_override**(name: StringName, constant: int) 🔗

Creates a local override for a theme constant with the specified `name`. Local overrides always take precedence when fetching theme items for the control. An override can be removed with remove_theme_constant_override.

See also get_theme_constant.

---

void **add_theme_font_override**(name: StringName, font: Font)

Creates a local override for a theme Font with the specified `name`. Local overrides always take precedence when fetching theme items for the control. An override can be removed with remove_theme_font_override.

See also get_theme_font.

---

void **add_theme_font_size_override**(name: StringName, font_size: int)

Creates a local override for a theme font size with the specified `name`. Local overrides always take precedence when fetching theme items for the control. An override can be removed with remove_theme_font_size_override.

See also get_theme_font_size.

---

void **add_theme_icon_override**(name: StringName, texture: Texture2D)

Creates a local override for a theme icon with the specified `name`. Local overrides always take precedence when fetching theme items for the control. An override can be removed with remove_theme_icon_override.

See also get_theme_icon.

---

void **add_theme_stylebox_override**(name: StringName, stylebox: StyleBox)

Creates a local override for a theme StyleBox with the specified `name`. Local overrides always take precedence when fetching theme items for the control. An override can be removed with remove_theme_stylebox_override.

See also get_theme_stylebox.

Example of modifying a property in a StyleBox by duplicating it:

GDScript

```
# The snippet below assumes the child node MyButton has a
StyleBoxFlat assigned.
# Resources are shared across instances, so we need to duplicate it
# to avoid modifying the appearance of all other buttons.
var new_stylebox_normal =
$MyButton.get_theme_stylebox("normal").duplicate()
new_stylebox_normal.border_width_top = 3
new_stylebox_normal.border_color = Color(0, 1, 0.5)
$MyButton.add_theme_stylebox_override("normal", new_stylebox_normal)
# Remove the stylebox override.
$MyButton.remove_theme_stylebox_override("normal")
```

C#

```
// The snippet below assumes the child node MyButton has a
StyleBoxFlat assigned.
// Resources are shared across instances, so we need to duplicate it
// to avoid modifying the appearance of all other buttons.
StyleBoxFlat newStyleboxNormal = GetNode<Button>
("MyButton").GetThemeStylebox("normal").Duplicate() as StyleBoxFlat;
newStyleboxNormal.BorderWidthTop = 3;
newStyleboxNormal.BorderColor = new Color(0, 1, 0.5f);
GetNode<Button>("MyButton").AddThemeStyleboxOverride("normal",
newStyleboxNormal);
// Remove the stylebox override.
GetNode<Button>("MyButton").RemoveThemeStyleboxOverride("normal");
```

---

void **begin_bulk_theme_override**() 🔗

Prevents `*_theme_*_override` methods from emitting NOTIFICATION_THEME_CHANGED until end_bulk_theme_override is called.

---

void **end_bulk_theme_override**() 🔗

Ends a bulk theme override update. See begin_bulk_theme_override.

Control **find_next_valid_focus**() const

Finds the next (below in the tree) **Control** that can receive the focus.

Control **find_prev_valid_focus**() const

Finds the previous (above in the tree) **Control** that can receive the focus.

Control **find_valid_focus_neighbor**(side: Side) const

Finds the next **Control** that can receive the focus on the specified Side.

Note: This is different from get_focus_neighbor, which returns the path of a specified focus neighbor.

---

void **force_drag**(data: Variant, preview: Control)

Forces drag and bypasses _get_drag_data and set_drag_preview by passing `data` and `preview`. Drag will start even if the mouse is neither over nor pressed on this control.

The methods _can_drop_data and _drop_data must be implemented on controls that want to receive drop data.

---

float **get_anchor**(side: Side) const

Returns the anchor for the specified Side. A getter method for anchor_bottom, anchor_left, anchor_right and anchor_top.

Vector2 **get_begin**() const

Returns offset_left and offset_top. See also position.

Vector2 **get_combined_minimum_size**() const

Returns combined minimum size from custom_minimum_size and get_minimum_size.

CursorShape **get_cursor_shape**(position: Vector2 = Vector2(0, 0)) const 🔗

Returns the mouse cursor shape the control displays on mouse hover. See CursorShape.

Vector2 **get_end**() const 🔗

Returns offset_right and offset_bottom.

NodePath **get_focus_neighbor**(side: Side) const 🔗

Returns the focus neighbor for the specified Side. A getter method for focus_neighbor_bottom, focus_neighbor_left, focus_neighbor_right and focus_neighbor_top.

Note: To find the next **Control** on the specific Side, even if a neighbor is not assigned, use find_valid_focus_neighbor.

Rect2 **get_global_rect**() const 🔗

Returns the position and size of the control relative to the containing canvas. See global_position and size.

Note: If the node itself or any parent CanvasItem between the node and the canvas have a non default rotation or skew, the resulting size is likely not meaningful.

Note: Setting Viewport.gui_snap_controls_to_pixels to `true` can lead to rounding inaccuracies between the displayed control and the returned Rect2.

Vector2 **get_minimum_size**() const

Returns the minimum size for this control. See custom_minimum_size.

---

float **get_offset**(offset: Side) const

Returns the offset for the specified Side. A getter method for offset_bottom, offset_left, offset_right and offset_top.

Vector2 **get_parent_area_size**() const

Returns the width/height occupied in the parent control.

Control **get_parent_control**() const

Returns the parent control node.

Rect2 **get_rect**() const

Returns the position and size of the control in the coordinate system of the containing node. See position, scale and size.

Note: If rotation is not the default rotation, the resulting size is not meaningful.

Note: Setting Viewport.gui_snap_controls_to_pixels to `true` can lead to rounding inaccuracies between the displayed control and the returned Rect2.

Vector2 **get_screen_position**() const

Returns the position of this **Control** in global screen coordinates (i.e. taking window position into account). Mostly useful for editor plugins.

Equals to global_position if the window is embedded (see Viewport.gui_embed_subwindows).

Example usage for showing a popup:

```
popup_menu.position = get_screen_position() +
get_local_mouse_position()
popup_menu.reset_size()
popup_menu.popup()
```

Color **get_theme_color**(name: StringName, theme_type: StringName = &"") const

Returns a Color from the first matching Theme in the tree if that Theme has a color item with the specified `name` and `theme_type`. If `theme_type` is omitted the class name of the current control is used as the type, or theme_type_variation if it is defined. If the type is a class name its parent classes are also checked, in order of inheritance. If the type is a variation its base types are checked, in order of dependency, then the control's class name and its parent classes are checked.

For the current control its local overrides are considered first (see add_theme_color_override), then its assigned theme. After the current control, each parent control and its assigned theme are considered; controls without a theme assigned are skipped. If no matching Theme is found in the tree, the custom project Theme (see ProjectSettings.gui/theme/custom) and the default Theme are used (see ThemeDB).

GDScript

```
func _ready():
    # Get the font color defined for the current Control's class, if
it exists.
    modulate = get_theme_color("font_color")
    # Get the font color defined for the Button class.
    modulate = get_theme_color("font_color", "Button")
```

C#

```
public override void _Ready()
{
    // Get the font color defined for the current Control's class,
if it exists.
    Modulate = GetThemeColor("font_color");
    // Get the font color defined for the Button class.
    Modulate = GetThemeColor("font_color", "Button");
}
```

---

int **get_theme_constant**(name: StringName, theme_type: StringName = &"") const

Returns a constant from the first matching Theme in the tree if that Theme has a constant item with the specified `name` and `theme_type`.

See get_theme_color for details.

---

float **get_theme_default_base_scale**() const

Returns the default base scale value from the first matching Theme in the tree if that Theme has a valid Theme.default_base_scale value.

See get_theme_color for details.

Font **get_theme_default_font**() const

Returns the default font from the first matching Theme in the tree if that Theme has a valid Theme.default_font value.

See get_theme_color for details.

---

int **get_theme_default_font_size**() const

Returns the default font size value from the first matching Theme in the tree if that Theme has a valid Theme.default_font_size value.

See get_theme_color for details.

Font **get_theme_font**(name: StringName, theme_type: StringName = &"") const

Returns a Font from the first matching Theme in the tree if that Theme has a font item with the specified `name` and `theme_type`.

See get_theme_color for details.

---

int **get_theme_font_size**(name: StringName, theme_type: StringName = &"") const

Returns a font size from the first matching Theme in the tree if that Theme has a font size item with the specified `name` and `theme_type`.

See get_theme_color for details.

Texture2D **get_theme_icon**(name: StringName, theme_type: StringName = &"") const

Returns an icon from the first matching Theme in the tree if that Theme has an icon item with the specified `name` and `theme_type`.

See get_theme_color for details.

StyleBox **get_theme_stylebox**(name: StringName, theme_type: StringName = &"") const

Returns a StyleBox from the first matching Theme in the tree if that Theme has a stylebox item with the specified `name` and `theme_type`.

See get_theme_color for details.

String **get_tooltip**(at_position: Vector2 = Vector2(0, 0)) const

Returns the tooltip text for the position `at_position` in control's local coordinates, which will typically appear when the cursor is resting over this control. By default, it returns tooltip_text.

This method can be overridden to customize its behavior. See _get_tooltip.

Note: If this method returns an empty String, no tooltip is displayed.

---

void **grab_click_focus**() 🔗

Creates an InputEventMouseButton that attempts to click the control. If the event is received, the control acquires focus.

GDScript

```
func _process(delta):
    grab_click_focus() # When clicking another Control node, this
node will be clicked instead.
```

C#

```
public override void _Process(double delta)
{
    GrabClickFocus(); // When clicking another Control node, this
node will be clicked instead.
}
```

---

void **grab_focus**() 🔗

Steal the focus from another control and become the focused control (see focus_mode).

Note: Using this method together with Callable.call_deferred makes it more reliable, especially when called inside Node._ready.

---

bool **has_focus**() const 🔗

Returns `true` if this is the current focused control. See focus_mode.

---

bool **has_theme_color**(name: StringName, theme_type: StringName = &"") const

Returns `true` if there is a matching Theme in the tree that has a color item with the specified `name` and `theme_type`.

See get_theme_color for details.

---

bool **has_theme_color_override**(name: StringName) const

Returns `true` if there is a local override for a theme Color with the specified `name` in this **Control** node.

See add_theme_color_override.

---

bool **has_theme_constant**(name: StringName, theme_type: StringName = &"") const

Returns `true` if there is a matching Theme in the tree that has a constant item with the specified `name` and `theme_type`.

See get_theme_color for details.

---

bool **has_theme_constant_override**(name: StringName) const

Returns `true` if there is a local override for a theme constant with the specified `name` in this **Control** node.

See add_theme_constant_override.

---

bool **has_theme_font**(name: StringName, theme_type: StringName = &"") const

Returns `true` if there is a matching Theme in the tree that has a font item with the specified `name` and `theme_type`.

See get_theme_color for details.

---

bool **has_theme_font_override**(name: StringName) const 🔗

Returns `true` if there is a local override for a theme Font with the specified `name` in this **Control** node.

See add_theme_font_override.

---

bool **has_theme_font_size**(name: StringName, theme_type: StringName = &"") const 🔗

Returns `true` if there is a matching Theme in the tree that has a font size item with the specified `name` and `theme_type`.

See get_theme_color for details.

---

bool **has_theme_font_size_override**(name: StringName) const 🔗

Returns `true` if there is a local override for a theme font size with the specified `name` in this **Control** node.

See add_theme_font_size_override.

---

bool **has_theme_icon**(name: StringName, theme_type: StringName = &"") const 🔗

Returns `true` if there is a matching Theme in the tree that has an icon item with the specified `name` and `theme_type`.

See get_theme_color for details.

---

bool **has_theme_icon_override**(name: StringName) const 🔗

Returns `true` if there is a local override for a theme icon with the specified `name` in this **Control** node.

See add_theme_icon_override.

---

bool **has_theme_stylebox**(name: StringName, theme_type: StringName = &"") const

Returns `true` if there is a matching Theme in the tree that has a stylebox item with the specified `name` and `theme_type`.

See get_theme_color for details.

---

bool **has_theme_stylebox_override**(name: StringName) const

Returns `true` if there is a local override for a theme StyleBox with the specified `name` in this **Control** node.

See add_theme_stylebox_override.

---

bool **is_drag_successful**() const

Returns `true` if a drag operation is successful. Alternative to Viewport.gui_is_drag_successful.

Best used with Node.NOTIFICATION_DRAG_END.

---

bool **is_layout_rtl**() const

Returns `true` if layout is right-to-left.

---

void **release_focus**()

Give up the focus. No other control will be able to receive input.

---

void **remove_theme_color_override**(name: StringName) 🔗

Removes a local override for a theme Color with the specified `name` previously added by add_theme_color_override or via the Inspector dock.

void **remove_theme_constant_override**(name: StringName) 🔗

Removes a local override for a theme constant with the specified `name` previously added by add_theme_constant_override or via the Inspector dock.

void **remove_theme_font_override**(name: StringName) 🔗

Removes a local override for a theme Font with the specified `name` previously added by add_theme_font_override or via the Inspector dock.

void **remove_theme_font_size_override**(name: StringName) 🔗

Removes a local override for a theme font size with the specified `name` previously added by add_theme_font_size_override or via the Inspector dock.

void **remove_theme_icon_override**(name: StringName) 🔗

Removes a local override for a theme icon with the specified `name` previously added by add_theme_icon_override or via the Inspector dock.

void **remove_theme_stylebox_override**(name: StringName) 🔗

Removes a local override for a theme StyleBox with the specified `name` previously added by add_theme_stylebox_override or via the Inspector dock.

void **reset_size**() 🔗

Resets the size to get_combined_minimum_size. This is equivalent to calling `set_size(Vector2())` (or any size below the minimum).

---

void **set_anchor**(side: Side, anchor: float, keep_offset: bool = false, push_opposite_anchor: bool = true) 🔗

Sets the anchor for the specified Side to `anchor`. A setter method for anchor_bottom, anchor_left, anchor_right and anchor_top.

If `keep_offset` is `true`, offsets aren't updated after this operation.

If `push_opposite_anchor` is `true` and the opposite anchor overlaps this anchor, the opposite one will have its value overridden. For example, when setting left anchor to 1 and the right anchor has value of 0.5, the right anchor will also get value of 1. If `push_opposite_anchor` was `false`, the left anchor would get value 0.5.

---

void **set_anchor_and_offset**(side: Side, anchor: float, offset: float, push_opposite_anchor: bool = false) 🔗

Works the same as set_anchor, but instead of `keep_offset` argument and automatic update of offset, it allows to set the offset yourself (see set_offset).

---

void **set_anchors_and_offsets_preset**(preset: LayoutPreset, resize_mode: LayoutPresetMode = 0, margin: int = 0) 🔗

Sets both anchor preset and offset preset. See set_anchors_preset and set_offsets_preset.

---

void **set_anchors_preset**(preset: LayoutPreset, keep_offsets: bool = false) 🔗

Sets the anchors to a `preset` from LayoutPreset enum. This is the code equivalent to using the Layout menu in the 2D editor.

If `keep_offsets` is `true`, control's position will also be updated.

---

void **set_begin**(position: Vector2)

Sets offset_left and offset_top at the same time. Equivalent of changing position.

---

void **set_drag_forwarding**(drag_func: Callable, can_drop_func: Callable, drop_func: Callable)

Forwards the handling of this control's _get_drag_data, _can_drop_data and _drop_data virtual functions to delegate callables.

For each argument, if not empty, the delegate callable is used, otherwise the local (virtual) function is used.

The function format for each callable should be exactly the same as the virtual functions described above.

---

void **set_drag_preview**(control: Control)

Shows the given control at the mouse pointer. A good time to call this method is in _get_drag_data. The control must not be in the scene tree. You should not free the control, and you should not keep a reference to the control beyond the duration of the drag. It will be deleted automatically after the drag has ended.

GDScript

```
@export var color = Color(1, 0, 0, 1)

func _get_drag_data(position):
    # Use a control that is not in the tree
    var cpb = ColorPickerButton.new()
    cpb.color = color
    cpb.size = Vector2(50, 50)
    set_drag_preview(cpb)
    return color
```

C#

```
[Export]
private Color _color = new Color(1, 0, 0, 1);

public override Variant _GetDragData(Vector2 atPosition)
{
    // Use a control that is not in the tree
    var cpb = new ColorPickerButton();
    cpb.Color = _color;
    cpb.Size = new Vector2(50, 50);
    SetDragPreview(cpb);
    return _color;
}
```

---

void **set_end**(position: Vector2) 🔗

Sets offset_right and offset_bottom at the same time.

---

void **set_focus_neighbor**(side: Side, neighbor: NodePath) 🔗

Sets the focus neighbor for the specified Side to the **Control** at `neighbor` node path. A setter method for focus_neighbor_bottom, focus_neighbor_left, focus_neighbor_right and focus_neighbor_top.

---

void **set_global_position**(position: Vector2, keep_offsets: bool = false) 🔗

Sets the global_position to given `position`.

If `keep_offsets` is `true`, control's anchors will be updated instead of offsets.

---

void **set_offset**(side: Side, offset: float) 🔗

Sets the offset for the specified Side to `offset`. A setter method for offset_bottom, offset_left, offset_right and offset_top.

---

void **set_offsets_preset**(preset: LayoutPreset, resize_mode: LayoutPresetMode = 0, margin: int = 0)

Sets the offsets to a `preset` from LayoutPreset enum. This is the code equivalent to using the Layout menu in the 2D editor.

Use parameter `resize_mode` with constants from LayoutPresetMode to better determine the resulting size of the **Control**. Constant size will be ignored if used with presets that change size, e.g. PRESET_LEFT_WIDE.

Use parameter `margin` to determine the gap between the **Control** and the edges.

---

void **set_position**(position: Vector2, keep_offsets: bool = false)

Sets the position to given `position`.

If `keep_offsets` is `true`, control's anchors will be updated instead of offsets.

---

void **set_size**(size: Vector2, keep_offsets: bool = false)

Sets the size (see size).

If `keep_offsets` is `true`, control's anchors will be updated instead of offsets.

---

void **update_minimum_size**()

Invalidates the size cache in this node and in parent nodes up to top level. Intended to be used with get_minimum_size when the return value is changed. Setting custom_minimum_size directly calls this method automatically.

---

void **warp_mouse**(position: Vector2)

Moves the mouse cursor to `position`, relative to position of this **Control**.

Note: warp_mouse is only supported on Windows, macOS and Linux. It has no effect on Android, iOS and Web.

CPUParticles2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A CPU-based 2D particle emitter.

Description

CPU-based 2D particle node used to create a variety of particle systems and effects.

See also GPUParticles2D, which provides the same functionality with hardware acceleration, but may not run on older devices.

Tutorials

- Particle systems (2D)

Properties

int	amount	`8`
Curve	angle_curve	
float	angle_max	`0.0`
float	angle_min	`0.0`

Curve	angular_velocity_curve	
float	angular_velocity_max	`0.0`
float	angular_velocity_min	`0.0`
Curve	anim_offset_curve	
float	anim_offset_max	`0.0`
float	anim_offset_min	`0.0`
Curve	anim_speed_curve	
float	anim_speed_max	`0.0`
float	anim_speed_min	`0.0`
Color	color	`Color(1, 1, 1, 1)`
Gradient	color_initial_ramp	
Gradient	color_ramp	
Curve	damping_curve	

float	damping_max	`0.0`
float	damping_min	`0.0`
Vector2	direction	`Vector2(1, 0)`
DrawOrder	draw_order	`0`
PackedColorArray	emission_colors	
PackedVector2Array	emission_normals	
PackedVector2Array	emission_points	
Vector2	emission_rect_extents	
EmissionShape	emission_shape	`0`
float	emission_sphere_radius	
bool	emitting	`true`
float	explosiveness	`0.0`

int	fixed_fps	`0`
bool	fract_delta	`true`
Vector2	gravity	`Vector2(0, 980)`
Curve	hue_variation_curve	
float	hue_variation_max	`0.0`
float	hue_variation_min	`0.0`
float	initial_velocity_max	`0.0`
float	initial_velocity_min	`0.0`
float	lifetime	`1.0`
float	lifetime_randomness	`0.0`
Curve	linear_accel_curve	
float	linear_accel_max	`0.0`
float	linear_accel_min	`0.0`

bool	local_coords	`false`
bool	one_shot	`false`
Curve	orbit_velocity_curve	
float	orbit_velocity_max	`0.0`
float	orbit_velocity_min	`0.0`
bool	particle_flag_align_y	`false`
float	preprocess	`0.0`
Curve	radial_accel_curve	
float	radial_accel_max	`0.0`
float	radial_accel_min	`0.0`
float	randomness	`0.0`
Curve	scale_amount_curve	

float	scale_amount_max	`1.0`
float	scale_amount_min	`1.0`
Curve	scale_curve_x	
Curve	scale_curve_y	
float	speed_scale	`1.0`
bool	split_scale	`false`
float	spread	`45.0`
Curve	tangential_accel_curve	
float	tangential_accel_max	`0.0`
float	tangential_accel_min	`0.0`
Texture2D	texture	

Methods

void	convert_from_particles(particles: Node)
Curve	get_param_curve(param: Parameter) const
float	get_param_max(param: Parameter) const
float	get_param_min(param: Parameter) const
bool	get_particle_flag(particle_flag: ParticleFlags) const
void	restart()
void	set_param_curve(param: Parameter, curve: Curve)
void	set_param_max(param: Parameter, value: float)
void	set_param_min(param: Parameter, value: float)
void	set_particle_flag(particle_flag: ParticleFlags, enable: bool)

Signals

finished() 🔗

Emitted when all active particles have finished processing. When one_shot is disabled, particles will process continuously, so this is never emitted.

Enumerations

enum **DrawOrder**:

DrawOrder **DRAW_ORDER_INDEX** = `0`

Particles are drawn in the order emitted.

DrawOrder **DRAW_ORDER_LIFETIME** = `1`

Particles are drawn in order of remaining lifetime. In other words, the particle with the highest lifetime is drawn at the front.

---

enum **Parameter**:

Parameter **PARAM_INITIAL_LINEAR_VELOCITY** = `0`

Use with set_param_min, set_param_max, and set_param_curve to set initial velocity properties.

Parameter **PARAM_ANGULAR_VELOCITY** = `1`

Use with set_param_min, set_param_max, and set_param_curve to set angular velocity properties.

Parameter **PARAM_ORBIT_VELOCITY** = `2`

Use with set_param_min, set_param_max, and set_param_curve to set orbital velocity properties.

Parameter **PARAM_LINEAR_ACCEL** = `3`

Use with set_param_min, set_param_max, and set_param_curve to set linear acceleration properties.

Parameter **PARAM_RADIAL_ACCEL** = `4`

Use with set_param_min, set_param_max, and set_param_curve to set radial acceleration properties.

Parameter **PARAM_TANGENTIAL_ACCEL** = `5`

Use with set_param_min, set_param_max, and set_param_curve to set tangential acceleration properties.

Parameter **PARAM_DAMPING** = `6`

Use with set_param_min, set_param_max, and set_param_curve to set damping properties.

Parameter **PARAM_ANGLE** = `7`

Use with set_param_min, set_param_max, and set_param_curve to set angle properties.

Parameter **PARAM_SCALE** = `8`

Use with set_param_min, set_param_max, and set_param_curve to set scale properties.

Parameter **PARAM_HUE_VARIATION** = `9`

Use with set_param_min, set_param_max, and set_param_curve to set hue variation properties.

Parameter **PARAM_ANIM_SPEED** = `10`

Use with set_param_min, set_param_max, and set_param_curve to set animation speed properties.

Parameter **PARAM_ANIM_OFFSET** = `11`

Use with set_param_min, set_param_max, and set_param_curve to set animation offset properties.

Parameter **PARAM_MAX** = `12`

Represents the size of the Parameter enum.

---

enum **ParticleFlags**: 🔗

ParticleFlags **PARTICLE_FLAG_ALIGN_Y_TO_VELOCITY** = `0`

Use with set_particle_flag to set particle_flag_align_y.

ParticleFlags **PARTICLE_FLAG_ROTATE_Y** = `1`

Present for consistency with 3D particle nodes, not used in 2D.

ParticleFlags **PARTICLE_FLAG_DISABLE_Z** = `2`

Present for consistency with 3D particle nodes, not used in 2D.

ParticleFlags **PARTICLE_FLAG_MAX** = `3`

Represents the size of the ParticleFlags enum.

---

enum **EmissionShape**: 🔗

EmissionShape **EMISSION_SHAPE_POINT** = `0`

All particles will be emitted from a single point.

EmissionShape **EMISSION_SHAPE_SPHERE** = `1`

Particles will be emitted in the volume of a sphere flattened to two dimensions.

EmissionShape **EMISSION_SHAPE_SPHERE_SURFACE** = `2`

Particles will be emitted on the surface of a sphere flattened to two dimensions.

EmissionShape **EMISSION_SHAPE_RECTANGLE** = `3`

Particles will be emitted in the area of a rectangle.

EmissionShape **EMISSION_SHAPE_POINTS** = `4`

Particles will be emitted at a position chosen randomly among emission_points. Particle color will be modulated by emission_colors.

EmissionShape **EMISSION_SHAPE_DIRECTED_POINTS** = `5`

Particles will be emitted at a position chosen randomly among emission_points. Particle velocity and rotation will be set based on emission_normals. Particle color will be modulated by emission_colors.

EmissionShape **EMISSION_SHAPE_MAX** = `6`

Represents the size of the EmissionShape enum.

Property Descriptions

int **amount** = `8` 🔗

- void **set_amount**(value: int)
- int **get_amount**()

Number of particles emitted in one emission cycle.

Curve **angle_curve**

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's rotation will be animated along this Curve.

---

float **angle_max** = `0.0`

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum initial rotation applied to each particle, in degrees.

---

float **angle_min** = `0.0`

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of angle_max.

Curve **angular_velocity_curve**

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's angular velocity will vary along this Curve.

---

float **angular_velocity_max** = `0.0`

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum initial angular velocity (rotation speed) applied to each particle in *degrees* per second.

---

float **angular_velocity_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of angular_velocity_max.

Curve **anim_offset_curve** 🔗

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's animation offset will vary along this Curve.

---

float **anim_offset_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum animation offset that corresponds to frame index in the texture. `0` is the first frame, `1` is the last one. See CanvasItemMaterial.particles_animation.

---

float **anim_offset_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of anim_offset_max.

Curve **anim_speed_curve** 🔗

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's animation speed will vary along this Curve.

---

float **anim_speed_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum particle animation speed. Animation speed of `1` means that the particles will make full `0` to `1` offset cycle during lifetime, `2` means `2` cycles etc.

With animation speed greater than `1`, remember to enable CanvasItemMaterial.particles_anim_loop property if you want the animation to repeat.

---

float **anim_speed_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of anim_speed_max.

Color **color** = `Color(1, 1, 1, 1)` 🔗

- void **set_color**(value: Color)
- Color **get_color**()

Each particle's initial color. If texture is defined, it will be multiplied by this color.

Gradient **color_initial_ramp** 🔗

- void **set_color_initial_ramp**(value: Gradient)
- Gradient **get_color_initial_ramp**()

Each particle's initial color will vary along this GradientTexture1D (multiplied with color).

Gradient **color_ramp** 🔗

- void **set_color_ramp**(value: Gradient)
- Gradient **get_color_ramp**()

Each particle's color will vary along this Gradient (multiplied with color).

Curve **damping_curve** 🔗

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Damping will vary along this Curve.

---

float **damping_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

The maximum rate at which particles lose velocity. For example value of `100` means that the particle will go from `100` velocity to `0` in

`1` second.

---

float **damping_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of damping_max.

Vector2 **direction** = `Vector2(1, 0)` 🔗

- void **set_direction**(value: Vector2)
- Vector2 **get_direction**()

Unit vector specifying the particles' emission direction.

DrawOrder **draw_order** = `0` 🔗

- void **set_draw_order**(value: DrawOrder)
- DrawOrder **get_draw_order**()

Particle draw order. Uses DrawOrder values.

PackedColorArray **emission_colors** 🔗

- void **set_emission_colors**(value: PackedColorArray)
- PackedColorArray **get_emission_colors**()

Sets the Colors to modulate particles by when using EMISSION_SHAPE_POINTS or EMISSION_SHAPE_DIRECTED_POINTS.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedColorArray for more details.

PackedVector2Array **emission_normals** 🔗

- void **set_emission_normals**(value: PackedVector2Array)
- PackedVector2Array **get_emission_normals**()

Sets the direction the particles will be emitted in when using EMISSION_SHAPE_DIRECTED_POINTS.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

PackedVector2Array **emission_points** 🔗

- void **set_emission_points**(value: PackedVector2Array)
- PackedVector2Array **get_emission_points**()

Sets the initial positions to spawn particles when using EMISSION_SHAPE_POINTS or EMISSION_SHAPE_DIRECTED_POINTS.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

Vector2 **emission_rect_extents** 🔗

- void **set_emission_rect_extents**(value: Vector2)
- Vector2 **get_emission_rect_extents**()

The rectangle's extents if emission_shape is set to EMISSION_SHAPE_RECTANGLE.

EmissionShape **emission_shape** = `0` 🔗

- void **set_emission_shape**(value: EmissionShape)
- EmissionShape **get_emission_shape**()

Particles will be emitted inside this region. See EmissionShape for possible values.

---

float **emission_sphere_radius** 🔗

- void **set_emission_sphere_radius**(value: float)
- float **get_emission_sphere_radius**()

The sphere's radius if emission_shape is set to EMISSION_SHAPE_SPHERE.

---

bool **emitting** = `true` 🔗

- void **set_emitting**(value: bool)
- bool **is_emitting**()

If `true`, particles are being emitted. emitting can be used to start and stop particles from emitting. However, if one_shot is `true` setting emitting to `true` will not restart the emission cycle until after all active particles finish processing. You can use the finished signal to be notified once all active particles finish processing.

---

float **explosiveness** = `0.0` 🔗

- void **set_explosiveness_ratio**(value: float)

- float **get_explosiveness_ratio**()

How rapidly particles in an emission cycle are emitted. If greater than `0`, there will be a gap in emissions before the next cycle begins.

---

int **fixed_fps** = `0`

- void **set_fixed_fps**(value: int)
- int **get_fixed_fps**()

The particle system's frame rate is fixed to a value. For example, changing the value to 2 will make the particles render at 2 frames per second. Note this does not slow down the simulation of the particle system itself.

---

bool **fract_delta** = `true`

- void **set_fractional_delta**(value: bool)
- bool **get_fractional_delta**()

If `true`, results in fractional delta calculation which has a smoother particles display effect.

Vector2 **gravity** = `Vector2(0, 980)`

- void **set_gravity**(value: Vector2)
- Vector2 **get_gravity**()

Gravity applied to every particle.

Curve **hue_variation_curve**

- void **set_param_curve**(param: Parameter, curve: Curve)

- Curve **get_param_curve**(param: Parameter) const

Each particle's hue will vary along this Curve.

float **hue_variation_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum initial hue variation applied to each particle. It will shift the particle color's hue.

float **hue_variation_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of hue_variation_max.

float **initial_velocity_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum initial velocity magnitude for each particle. Direction comes from direction and spread.

float **initial_velocity_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of initial_velocity_max.

---

float **lifetime** = `1.0`

- void **set_lifetime**(value: float)
- float **get_lifetime**()

Amount of time each particle will exist.

---

float **lifetime_randomness** = `0.0`

- void **set_lifetime_randomness**(value: float)
- float **get_lifetime_randomness**()

Particle lifetime randomness ratio.

Curve **linear_accel_curve**

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's linear acceleration will vary along this Curve.

---

float **linear_accel_max** = `0.0`

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum linear acceleration applied to each particle in the direction of motion.

---

float **linear_accel_min** = `0.0`

- void **set_param_min**(param: Parameter, value: float)

- float **get_param_min**(param: Parameter) const

Minimum equivalent of linear_accel_max.

---

bool **local_coords** = `false` 🔗

- void **set_use_local_coordinates**(value: bool)
- bool **get_use_local_coordinates**()

If `true`, particles use the parent node's coordinate space (known as local coordinates). This will cause particles to move and rotate along the **CPUParticles2D** node (and its parents) when it is moved or rotated. If `false`, particles use global coordinates; they will not move or rotate along the **CPUParticles2D** node (and its parents) when it is moved or rotated.

---

bool **one_shot** = `false` 🔗

- void **set_one_shot**(value: bool)
- bool **get_one_shot**()

If `true`, only one emission cycle occurs. If set `true` during a cycle, emission will stop at the cycle's end.

Curve **orbit_velocity_curve** 🔗

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's orbital velocity will vary along this Curve.

---

float **orbit_velocity_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum orbital velocity applied to each particle. Makes the particles circle around origin. Specified in number of full rotations around origin per second.

---

float **orbit_velocity_min** = `0.0`

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of orbit_velocity_max.

---

bool **particle_flag_align_y** = `false`

- void **set_particle_flag**(particle_flag: ParticleFlags, enable: bool)
- bool **get_particle_flag**(particle_flag: ParticleFlags) const

Align Y axis of particle with the direction of its velocity.

---

float **preprocess** = `0.0`

- void **set_pre_process_time**(value: float)
- float **get_pre_process_time**()

Particle system starts as if it had already run for this many seconds.

Curve **radial_accel_curve**

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's radial acceleration will vary along this Curve.

---

float **radial_accel_max** = `0.0`

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum radial acceleration applied to each particle. Makes particle accelerate away from the origin or towards it if negative.

---

float **radial_accel_min** = `0.0`

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of radial_accel_max.

---

float **randomness** = `0.0`

- void **set_randomness_ratio**(value: float)
- float **get_randomness_ratio**()

Emission lifetime randomness ratio.

Curve **scale_amount_curve**

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's scale will vary along this Curve.

---

float **scale_amount_max** = `1.0`

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum initial scale applied to each particle.

float **scale_amount_min** = `1.0`

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of scale_amount_max.

Curve **scale_curve_x**

- void **set_scale_curve_x**(value: Curve)
- Curve **get_scale_curve_x**()

Each particle's horizontal scale will vary along this Curve.

split_scale must be enabled.

Curve **scale_curve_y**

- void **set_scale_curve_y**(value: Curve)
- Curve **get_scale_curve_y**()

Each particle's vertical scale will vary along this Curve.

split_scale must be enabled.

float **speed_scale** = `1.0`

- void **set_speed_scale**(value: float)

- float **get_speed_scale**()

Particle system's running speed scaling ratio. A value of `0` can be used to pause the particles.

---

bool **split_scale** = `false` 🔗

- void **set_split_scale**(value: bool)
- bool **get_split_scale**()

If `true`, the scale curve will be split into x and y components. See scale_curve_x and scale_curve_y.

---

float **spread** = `45.0` 🔗

- void **set_spread**(value: float)
- float **get_spread**()

Each particle's initial direction range from `+spread` to `-spread` degrees.

Curve **tangential_accel_curve** 🔗

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's tangential acceleration will vary along this Curve.

---

float **tangential_accel_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum tangential acceleration applied to each particle. Tangential acceleration is perpendicular to the particle's velocity giving the particles a swirling motion.

---

float **tangential_accel_min** = `0.0`

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of tangential_accel_max.

Texture2D **texture**

- void **set_texture**(value: Texture2D)
- Texture2D **get_texture**()

Particle texture. If `null`, particles will be squares.

Method Descriptions

void **convert_from_particles**(particles: Node)

Sets this node's properties to match a given GPUParticles2D node with an assigned ParticleProcessMaterial.

Curve **get_param_curve**(param: Parameter) const

Returns the Curve of the parameter specified by Parameter.

---

float **get_param_max**(param: Parameter) const

Returns the maximum value range for the given parameter.

---

float **get_param_min**(param: Parameter) const

Returns the minimum value range for the given parameter.

---

bool **get_particle_flag**(particle_flag: ParticleFlags) const

Returns the enabled state of the given flag (see ParticleFlags for options).

---

void **restart**()

Restarts the particle emitter.

---

void **set_param_curve**(param: Parameter, curve: Curve)

Sets the Curve of the parameter specified by Parameter.

---

void **set_param_max**(param: Parameter, value: float)

Sets the maximum value for the given parameter.

---

void **set_param_min**(param: Parameter, value: float)

Sets the minimum value for the given parameter.

---

void **set_particle_flag**(particle_flag: ParticleFlags, enable: bool)

Enables or disables the given flag (see ParticleFlags for options).

CPUParticles3D

Inherits: GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A CPU-based 3D particle emitter.

Description

CPU-based 3D particle node used to create a variety of particle systems and effects.

See also GPUParticles3D, which provides the same functionality with hardware acceleration, but may not run on older devices.

Tutorials

- Particle systems (3D)

Properties

int	amount	`8`
Curve	angle_curve	
float	angle_max	`0.0`
float	angle_min	`0.0`

Curve	angular_velocity_curve	
float	angular_velocity_max	`0.0`
float	angular_velocity_min	`0.0`
Curve	anim_offset_curve	
float	anim_offset_max	`0.0`
float	anim_offset_min	`0.0`
Curve	anim_speed_curve	
float	anim_speed_max	`0.0`
float	anim_speed_min	`0.0`
Color	color	`Color(1, 1, 1, 1)`
Gradient	color_initial_ramp	
Gradient	color_ramp	
Curve	damping_curve	

float	damping_max	`0.0`
float	damping_min	`0.0`
Vector3	direction	`Vector3(1, 0, 0)`
DrawOrder	draw_order	`0`
Vector3	emission_box_extents	
PackedColorArray	emission_colors	`PackedColorArray()`
PackedVector3Array	emission_normals	
PackedVector3Array	emission_points	
Vector3	emission_ring_axis	
float	emission_ring_height	
float	emission_ring_inner_radius	
float	emission_ring_radius	
EmissionShape	emission_shape	`0`

float	emission_sphere_radius	
bool	emitting	`true`
float	explosiveness	`0.0`
int	fixed_fps	`0`
float	flatness	`0.0`
bool	fract_delta	`true`
Vector3	gravity	`Vector3(0, -9.8, 0)`
Curve	hue_variation_curve	
float	hue_variation_max	`0.0`
float	hue_variation_min	`0.0`
float	initial_velocity_max	`0.0`
float	initial_velocity_min	`0.0`

float	lifetime	`1.0`
float	lifetime_randomness	`0.0`
Curve	linear_accel_curve	
float	linear_accel_max	`0.0`
float	linear_accel_min	`0.0`
bool	local_coords	`false`
Mesh	mesh	
bool	one_shot	`false`
Curve	orbit_velocity_curve	
float	orbit_velocity_max	
float	orbit_velocity_min	
bool	particle_flag_align_y	`false`
bool	particle_flag_disable_z	`false`

bool	particle_flag_rotate_y	`false`
float	preprocess	`0.0`
Curve	radial_accel_curve	
float	radial_accel_max	`0.0`
float	radial_accel_min	`0.0`
float	randomness	`0.0`
Curve	scale_amount_curve	
float	scale_amount_max	`1.0`
float	scale_amount_min	`1.0`
Curve	scale_curve_x	
Curve	scale_curve_y	
Curve	scale_curve_z	
float	speed_scale	`1.0`

bool	split_scale	`false`
float	spread	`45.0`
Curve	tangential_accel_curve	
float	tangential_accel_max	`0.0`
float	tangential_accel_min	`0.0`
AABB	visibility_aabb	`AABB(0, 0, 0, 0, 0, 0)`

Methods

void	convert_from_particles(particles: Node)
Curve	get_param_curve(param: Parameter) const
float	get_param_max(param: Parameter) const
float	get_param_min(param: Parameter) const
bool	get_particle_flag(particle_flag: ParticleFlags) const

void	restart()
void	set_param_curve(param: Parameter, curve: Curve)
void	set_param_max(param: Parameter, value: float)
void	set_param_min(param: Parameter, value: float)
void	set_particle_flag(particle_flag: ParticleFlags, enable: bool)

Signals

finished() 🔗

Emitted when all active particles have finished processing. When one_shot is disabled, particles will process continuously, so this is never emitted.

Enumerations

enum **DrawOrder**: 🔗

DrawOrder **DRAW_ORDER_INDEX** = `0`

Particles are drawn in the order emitted.

DrawOrder **DRAW_ORDER_LIFETIME** = `1`

Particles are drawn in order of remaining lifetime. In other words, the particle with the highest lifetime is drawn at the front.

DrawOrder **DRAW_ORDER_VIEW_DEPTH** = `2`

Particles are drawn in order of depth.

---

enum **Parameter**: 🔗

Parameter **PARAM_INITIAL_LINEAR_VELOCITY** = `0`

Use with set_param_min, set_param_max, and set_param_curve to set initial velocity properties.

Parameter **PARAM_ANGULAR_VELOCITY** = `1`

Use with set_param_min, set_param_max, and set_param_curve to set angular velocity properties.

Parameter **PARAM_ORBIT_VELOCITY** = `2`

Use with set_param_min, set_param_max, and set_param_curve to set orbital velocity properties.

Parameter **PARAM_LINEAR_ACCEL** = `3`

Use with set_param_min, set_param_max, and set_param_curve to set linear acceleration properties.

Parameter **PARAM_RADIAL_ACCEL** = `4`

Use with set_param_min, set_param_max, and set_param_curve to set radial acceleration properties.

Parameter **PARAM_TANGENTIAL_ACCEL** = `5`

Use with set_param_min, set_param_max, and set_param_curve to set tangential acceleration properties.

Parameter **PARAM_DAMPING** = `6`

Use with set_param_min, set_param_max, and set_param_curve to set damping properties.

Parameter **PARAM_ANGLE** = `7`

Use with set_param_min, set_param_max, and set_param_curve to set angle properties.

Parameter **PARAM_SCALE** = `8`

Use with set_param_min, set_param_max, and set_param_curve to set scale properties.

Parameter **PARAM_HUE_VARIATION** = `9`

Use with set_param_min, set_param_max, and set_param_curve to set hue variation properties.

Parameter **PARAM_ANIM_SPEED** = `10`

Use with set_param_min, set_param_max, and set_param_curve to set animation speed properties.

Parameter **PARAM_ANIM_OFFSET** = `11`

Use with set_param_min, set_param_max, and set_param_curve to set animation offset properties.

Parameter **PARAM_MAX** = `12`

Represents the size of the Parameter enum.

enum **ParticleFlags**: 🔗

ParticleFlags **PARTICLE_FLAG_ALIGN_Y_TO_VELOCITY** = `0`

Use with set_particle_flag to set particle_flag_align_y.

ParticleFlags **PARTICLE_FLAG_ROTATE_Y** = `1`

Use with set_particle_flag to set particle_flag_rotate_y.

ParticleFlags **PARTICLE_FLAG_DISABLE_Z** = `2`

Use with set_particle_flag to set particle_flag_disable_z.

ParticleFlags **PARTICLE_FLAG_MAX** = `3`

Represents the size of the ParticleFlags enum.

---

enum **EmissionShape**: 🔗

EmissionShape **EMISSION_SHAPE_POINT** = `0`

All particles will be emitted from a single point.

EmissionShape **EMISSION_SHAPE_SPHERE** = `1`

Particles will be emitted in the volume of a sphere.

EmissionShape **EMISSION_SHAPE_SPHERE_SURFACE** = `2`

Particles will be emitted on the surface of a sphere.

EmissionShape **EMISSION_SHAPE_BOX** = `3`

Particles will be emitted in the volume of a box.

EmissionShape **EMISSION_SHAPE_POINTS** = `4`

Particles will be emitted at a position chosen randomly among emission_points. Particle color will be modulated by emission_colors.

EmissionShape **EMISSION_SHAPE_DIRECTED_POINTS** = `5`

Particles will be emitted at a position chosen randomly among emission_points. Particle velocity and rotation will be set based on emission_normals. Particle color will be modulated by emission_colors.

EmissionShape **EMISSION_SHAPE_RING** = `6`

Particles will be emitted in a ring or cylinder.

EmissionShape **EMISSION_SHAPE_MAX** = `7`

Represents the size of the EmissionShape enum.

Property Descriptions

int **amount** = `8`

- void **set_amount**(value: int)
- int **get_amount**()

Number of particles emitted in one emission cycle.

Curve **angle_curve**

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's rotation will be animated along this Curve.

---

float **angle_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum angle.

---

float **angle_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum angle.

Curve **angular_velocity_curve** 🔗

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's angular velocity (rotation speed) will vary along this Curve over its lifetime.

---

float **angular_velocity_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum initial angular velocity (rotation speed) applied to each particle in *degrees* per second.

---

float **angular_velocity_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum initial angular velocity (rotation speed) applied to each particle in *degrees* per second.

Curve **anim_offset_curve**

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's animation offset will vary along this Curve.

---

float **anim_offset_max** = `0.0`

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum animation offset.

---

float **anim_offset_min** = `0.0`

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum animation offset.

Curve **anim_speed_curve**

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's animation speed will vary along this Curve.

---

float **anim_speed_max** = `0.0`

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum particle animation speed.

---

float **anim_speed_min** = `0.0`

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum particle animation speed.

Color **color** = `Color(1, 1, 1, 1)`

- void **set_color**(value: Color)
- Color **get_color**()

Each particle's initial color.

Note: color multiplies the particle mesh's vertex colors. To have a visible effect on a BaseMaterial3D, BaseMaterial3D.vertex_color_use_as_albedo *must* be `true`. For a ShaderMaterial, `ALBEDO *= COLOR.rgb;` must be inserted in the shader's `fragment()` function. Otherwise, color will have no visible effect.

Gradient **color_initial_ramp**

- void **set_color_initial_ramp**(value: Gradient)
- Gradient **get_color_initial_ramp**()

Each particle's initial color will vary along this GradientTexture1D (multiplied with color).

Note: color_initial_ramp multiplies the particle mesh's vertex colors. To have a visible effect on a BaseMaterial3D, BaseMaterial3D.vertex_color_use_as_albedo *must* be `true`. For a ShaderMaterial, `ALBEDO *= COLOR.rgb;` must be inserted in the shader's `fragment()` function. Otherwise, color_initial_ramp will have no visible effect.

Gradient **color_ramp**

- void **set_color_ramp**(value: Gradient)
- Gradient **get_color_ramp**()

Each particle's color will vary along this GradientTexture1D over its lifetime (multiplied with color).

Note: color_ramp multiplies the particle mesh's vertex colors. To have a visible effect on a BaseMaterial3D, BaseMaterial3D.vertex_color_use_as_albedo *must* be `true`. For a ShaderMaterial, `ALBEDO *= COLOR.rgb;` must be inserted in the shader's `fragment()` function. Otherwise, color_ramp will have no visible effect.

Curve **damping_curve**

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Damping will vary along this Curve.

---

float **damping_max** = `0.0`

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum damping.

---

float **damping_min** = `0.0`

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum damping.

Vector3 **direction** = `Vector3(1, 0, 0)`

- void **set_direction**(value: Vector3)
- Vector3 **get_direction**()

Unit vector specifying the particles' emission direction.

DrawOrder **draw_order** = `0`

- void **set_draw_order**(value: DrawOrder)
- DrawOrder **get_draw_order**()

Particle draw order. Uses DrawOrder values.

Vector3 **emission_box_extents**

- void **set_emission_box_extents**(value: Vector3)
- Vector3 **get_emission_box_extents**()

The rectangle's extents if emission_shape is set to EMISSION_SHAPE_BOX.

PackedColorArray **emission_colors** = `PackedColorArray()`

- void **set_emission_colors**(value: PackedColorArray)
- PackedColorArray **get_emission_colors**()

Sets the Colors to modulate particles by when using EMISSION_SHAPE_POINTS or EMISSION_SHAPE_DIRECTED_POINTS.

Note: emission_colors multiplies the particle mesh's vertex colors. To have a visible effect on a BaseMaterial3D, BaseMaterial3D.vertex_color_use_as_albedo *must* be `true`. For a ShaderMaterial, `ALBEDO *= COLOR.rgb;` must be inserted in the shader's `fragment()` function. Otherwise, emission_colors will have no visible effect.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedColorArray for more details.

PackedVector3Array **emission_normals** 🔗

- void **set_emission_normals**(value: PackedVector3Array)
- PackedVector3Array **get_emission_normals**()

Sets the direction the particles will be emitted in when using EMISSION_SHAPE_DIRECTED_POINTS.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector3Array for more details.

PackedVector3Array **emission_points** 🔗

- void **set_emission_points**(value: PackedVector3Array)
- PackedVector3Array **get_emission_points**()

Sets the initial positions to spawn particles when using EMISSION_SHAPE_POINTS or EMISSION_SHAPE_DIRECTED_POINTS.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector3Array for more details.

Vector3 **emission_ring_axis** 🔗

- void **set_emission_ring_axis**(value: Vector3)
- Vector3 **get_emission_ring_axis**()

The axis of the ring when using the emitter EMISSION_SHAPE_RING.

---

float **emission_ring_height** 🔗

- void **set_emission_ring_height**(value: float)
- float **get_emission_ring_height**()

The height of the ring when using the emitter EMISSION_SHAPE_RING.

---

float **emission_ring_inner_radius** 🔗

- void **set_emission_ring_inner_radius**(value: float)
- float **get_emission_ring_inner_radius**()

The inner radius of the ring when using the emitter EMISSION_SHAPE_RING.

---

float **emission_ring_radius** 🔗

- void **set_emission_ring_radius**(value: float)

- float **get_emission_ring_radius**()

The radius of the ring when using the emitter EMISSION_SHAPE_RING.

EmissionShape **emission_shape** = `0` 🔗

- void **set_emission_shape**(value: EmissionShape)
- EmissionShape **get_emission_shape**()

Particles will be emitted inside this region. See EmissionShape for possible values.

---

float **emission_sphere_radius** 🔗

- void **set_emission_sphere_radius**(value: float)
- float **get_emission_sphere_radius**()

The sphere's radius if EmissionShape is set to EMISSION_SHAPE_SPHERE.

---

bool **emitting** = `true` 🔗

- void **set_emitting**(value: bool)
- bool **is_emitting**()

If `true`, particles are being emitted. emitting can be used to start and stop particles from emitting. However, if one_shot is `true` setting emitting to `true` will not restart the emission cycle until after all active particles finish processing. You can use the finished signal to be notified once all active particles finish processing.

---

float **explosiveness** = `0.0` 🔗

- void **set_explosiveness_ratio**(value: float)
- float **get_explosiveness_ratio**()

How rapidly particles in an emission cycle are emitted. If greater than `0`, there will be a gap in emissions before the next cycle begins.

---

int **fixed_fps** = `0`

- void **set_fixed_fps**(value: int)
- int **get_fixed_fps**()

The particle system's frame rate is fixed to a value. For example, changing the value to 2 will make the particles render at 2 frames per second. Note this does not slow down the particle system itself.

---

float **flatness** = `0.0`

- void **set_flatness**(value: float)
- float **get_flatness**()

Amount of spread in Y/Z plane. A value of `1` restricts particles to X/Z plane.

---

bool **fract_delta** = `true`

- void **set_fractional_delta**(value: bool)
- bool **get_fractional_delta**()

If `true`, results in fractional delta calculation which has a smoother particles display effect.

Vector3 **gravity** = `Vector3(0, -9.8, 0)`

- void **set_gravity**(value: Vector3)
- Vector3 **get_gravity**()

Gravity applied to every particle.

Curve **hue_variation_curve** 🔗

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's hue will vary along this Curve.

---

float **hue_variation_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum hue variation.

---

float **hue_variation_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum hue variation.

---

float **initial_velocity_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum value of the initial velocity.

---

float **initial_velocity_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum value of the initial velocity.

---

float **lifetime** = `1.0` 🔗

- void **set_lifetime**(value: float)
- float **get_lifetime**()

Amount of time each particle will exist.

---

float **lifetime_randomness** = `0.0` 🔗

- void **set_lifetime_randomness**(value: float)
- float **get_lifetime_randomness**()

Particle lifetime randomness ratio.

Curve **linear_accel_curve** 🔗

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's linear acceleration will vary along this Curve.

---

float **linear_accel_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum linear acceleration.

---

float **linear_accel_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum linear acceleration.

---

bool **local_coords** = `false` 🔗

- void **set_use_local_coordinates**(value: bool)
- bool **get_use_local_coordinates**()

If `true`, particles use the parent node's coordinate space (known as local coordinates). This will cause particles to move and rotate along the **CPUParticles3D** node (and its parents) when it is moved or rotated. If `false`, particles use global coordinates; they will not move or rotate along the **CPUParticles3D** node (and its parents) when it is moved or rotated.

Mesh **mesh** 🔗

- void **set_mesh**(value: Mesh)
- Mesh **get_mesh**()

The Mesh used for each particle. If `null`, particles will be spheres.

---

bool **one_shot** = `false` 🔗

- void **set_one_shot**(value: bool)
- bool **get_one_shot**()

If `true`, only one emission cycle occurs. If set `true` during a cycle, emission will stop at the cycle's end.

Curve **orbit_velocity_curve** 🔗

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's orbital velocity will vary along this Curve.

---

float **orbit_velocity_max** 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum orbit velocity.

---

float **orbit_velocity_min** 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum orbit velocity.

---

bool **particle_flag_align_y** = `false` 🔗

- void **set_particle_flag**(particle_flag: ParticleFlags, enable: bool)
- bool **get_particle_flag**(particle_flag: ParticleFlags) const

Align Y axis of particle with the direction of its velocity.

---

bool **particle_flag_disable_z** = `false` 🔗

- void **set_particle_flag**(particle_flag: ParticleFlags, enable: bool)
- bool **get_particle_flag**(particle_flag: ParticleFlags) const

If `true`, particles will not move on the Z axis.

---

bool **particle_flag_rotate_y** = `false`

- void **set_particle_flag**(particle_flag: ParticleFlags, enable: bool)
- bool **get_particle_flag**(particle_flag: ParticleFlags) const

If `true`, particles rotate around Y axis by angle_min.

---

float **preprocess** = `0.0`

- void **set_pre_process_time**(value: float)
- float **get_pre_process_time**()

Particle system starts as if it had already run for this many seconds.

Curve **radial_accel_curve**

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's radial acceleration will vary along this Curve.

---

float **radial_accel_max** = `0.0`

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum radial acceleration.

---

float **radial_accel_min** = `0.0`

- void **set_param_min**(param: Parameter, value: float)

- float **get_param_min**(param: Parameter) const

Minimum radial acceleration.

---

float **randomness** = `0.0` 🔗

- void **set_randomness_ratio**(value: float)
- float **get_randomness_ratio**()

Emission lifetime randomness ratio.

Curve **scale_amount_curve** 🔗

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's scale will vary along this Curve.

---

float **scale_amount_max** = `1.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum scale.

---

float **scale_amount_min** = `1.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum scale.

Curve **scale_curve_x** 🔗

- void **set_scale_curve_x**(value: Curve)
- Curve **get_scale_curve_x**()

Curve for the scale over life, along the x axis.

Curve **scale_curve_y** 🔗

- void **set_scale_curve_y**(value: Curve)
- Curve **get_scale_curve_y**()

Curve for the scale over life, along the y axis.

Curve **scale_curve_z** 🔗

- void **set_scale_curve_z**(value: Curve)
- Curve **get_scale_curve_z**()

Curve for the scale over life, along the z axis.

---

float **speed_scale** = `1.0` 🔗

- void **set_speed_scale**(value: float)
- float **get_speed_scale**()

Particle system's running speed scaling ratio. A value of `0` can be used to pause the particles.

---

bool **split_scale** = `false` 🔗

- void **set_split_scale**(value: bool)
- bool **get_split_scale**()

If set to `true`, three different scale curves can be specified, one per scale axis.

---

float **spread** = `45.0` 🔗

- void **set_spread**(value: float)
- float **get_spread**()

Each particle's initial direction range from `+spread` to `-spread` degrees. Applied to X/Z plane and Y/Z planes.

Curve **tangential_accel_curve** 🔗

- void **set_param_curve**(param: Parameter, curve: Curve)
- Curve **get_param_curve**(param: Parameter) const

Each particle's tangential acceleration will vary along this Curve.

---

float **tangential_accel_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum tangent acceleration.

---

float **tangential_accel_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum tangent acceleration.

AABB **visibility_aabb** = `AABB(0, 0, 0, 0, 0, 0)` 🔗

- void **set_visibility_aabb**(value: AABB)
- AABB **get_visibility_aabb**()

The AABB that determines the node's region which needs to be visible on screen for the particle system to be active.

Grow the box if particles suddenly appear/disappear when the node enters/exits the screen. The AABB can be grown via code or with the **Particles → Generate AABB** editor tool.

Method Descriptions

void **convert_from_particles**(particles: Node)

Sets this node's properties to match a given GPUParticles3D node with an assigned ParticleProcessMaterial.

Curve **get_param_curve**(param: Parameter) const

Returns the Curve of the parameter specified by Parameter.

float **get_param_max**(param: Parameter) const

Returns the maximum value range for the given parameter.

float **get_param_min**(param: Parameter) const

Returns the minimum value range for the given parameter.

bool **get_particle_flag**(particle_flag: ParticleFlags) const

Returns the enabled state of the given particle flag (see ParticleFlags for options).

void **restart**()

Restarts the particle emitter.

---

void **set_param_curve**(param: Parameter, curve: Curve)

Sets the Curve of the parameter specified by Parameter.

---

void **set_param_max**(param: Parameter, value: float)

Sets the maximum value for the given parameter.

---

void **set_param_min**(param: Parameter, value: float)

Sets the minimum value for the given parameter.

---

void **set_particle_flag**(particle_flag: ParticleFlags, enable: bool)

Enables or disables the given particle flag (see ParticleFlags for options).

CSGBox3D

Inherits: CSGPrimitive3D **<** CSGShape3D **<** GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A CSG Box shape.

Description

This node allows you to create a box for use with the CSG system.

Note: CSG nodes are intended to be used for level prototyping. Creating CSG nodes has a significant CPU cost compared to creating a MeshInstance3D with a PrimitiveMesh. Moving a CSG node within another CSG node also has a significant CPU cost, so it should be avoided during gameplay.

Tutorials

- Prototyping levels with CSG

Properties

Material	material	
Vector3	size	`Vector3(1, 1, 1)`

Property Descriptions

Material **material** 🔗

- void **set_material**(value: Material)
- Material **get_material**()

The material used to render the box.

Vector3 **size** = `Vector3(1, 1, 1)` 🔗

- void **set_size**(value: Vector3)
- Vector3 **get_size**()

The box's width, height and depth.

CSGCombiner3D

Inherits: CSGShape3D **<** GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A CSG node that allows you to combine other CSG modifiers.

Description

For complex arrangements of shapes, it is sometimes needed to add structure to your CSG nodes. The CSGCombiner3D node allows you to create this structure. The node encapsulates the result of the CSG operations of its children. In this way, it is possible to do operations on one set of shapes that are children of one CSGCombiner3D node, and a set of separate operations on a second set of shapes that are children of a second CSGCombiner3D node, and then do an operation that takes the two end results as its input to create the final shape.

Note: CSG nodes are intended to be used for level prototyping. Creating CSG nodes has a significant CPU cost compared to creating a MeshInstance3D with a PrimitiveMesh. Moving a CSG node within another CSG node also has a significant CPU cost, so it should be avoided during gameplay.

Tutorials

- Prototyping levels with CSG

CSGCylinder3D

Inherits: CSGPrimitive3D **<** CSGShape3D **<** GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A CSG Cylinder shape.

Description

This node allows you to create a cylinder (or cone) for use with the CSG system.

Note: CSG nodes are intended to be used for level prototyping. Creating CSG nodes has a significant CPU cost compared to creating a MeshInstance3D with a PrimitiveMesh. Moving a CSG node within another CSG node also has a significant CPU cost, so it should be avoided during gameplay.

Tutorials

- Prototyping levels with CSG

Properties

bool	cone	`false`
float	height	`2.0`

Material	material	
float	radius	`0.5`
int	sides	`8`
bool	smooth_faces	`true`

Property Descriptions

bool **cone** = `false`

- void **set_cone**(value: bool)
- bool **is_cone**()

If `true` a cone is created, the radius will only apply to one side.

---

float **height** = `2.0`

- void **set_height**(value: float)
- float **get_height**()

The height of the cylinder.

Material **material**

- void **set_material**(value: Material)
- Material **get_material**()

The material used to render the cylinder.

---

float **radius** = `0.5` 🔗

- void **set_radius**(value: float)
- float **get_radius**()

The radius of the cylinder.

---

int **sides** = `8` 🔗

- void **set_sides**(value: int)
- int **get_sides**()

The number of sides of the cylinder, the higher this number the more detail there will be in the cylinder.

---

bool **smooth_faces** = `true` 🔗

- void **set_smooth_faces**(value: bool)
- bool **get_smooth_faces**()

If `true` the normals of the cylinder are set to give a smooth effect making the cylinder seem rounded. If `false` the cylinder will have a flat shaded look.

CSGMesh3D

Inherits: CSGPrimitive3D **<** CSGShape3D **<** GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A CSG Mesh shape that uses a mesh resource.

Description

This CSG node allows you to use any mesh resource as a CSG shape, provided it is closed, does not self-intersect, does not contain internal faces and has no edges that connect to more than two faces. See also CSGPolygon3D for drawing 2D extruded polygons to be used as CSG nodes.

Note: CSG nodes are intended to be used for level prototyping. Creating CSG nodes has a significant CPU cost compared to creating a MeshInstance3D with a PrimitiveMesh. Moving a CSG node within another CSG node also has a significant CPU cost, so it should be avoided during gameplay.

Tutorials

- Prototyping levels with CSG

Properties

Material	material

Mesh	mesh

Property Descriptions

Material **material** 🔗

- void **set_material**(value: Material)
- Material **get_material**()

The Material used in drawing the CSG shape.

Mesh **mesh** 🔗

- void **set_mesh**(value: Mesh)
- Mesh **get_mesh**()

The Mesh resource to use as a CSG shape.

Note: When using an ArrayMesh, all vertex attributes except Mesh.ARRAY_VERTEX, Mesh.ARRAY_NORMAL and Mesh.ARRAY_TEX_UV are left unused. Only Mesh.ARRAY_VERTEX and Mesh.ARRAY_TEX_UV will be passed to the GPU.

Mesh.ARRAY_NORMAL is only used to determine which faces require the use of flat shading. By default, CSGMesh will ignore the mesh's vertex normals, recalculate them for each vertex and use a smooth shader. If a flat shader is required for a face, ensure that all vertex normals of the face are approximately equal.

CSGPolygon3D

Inherits: CSGPrimitive3D **<** CSGShape3D **<** GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

Extrudes a 2D polygon shape to create a 3D mesh.

Description

An array of 2D points is extruded to quickly and easily create a variety of 3D meshes. See also CSGMesh3D for using 3D meshes as CSG nodes.

Note: CSG nodes are intended to be used for level prototyping. Creating CSG nodes has a significant CPU cost compared to creating a MeshInstance3D with a PrimitiveMesh. Moving a CSG node within another CSG node also has a significant CPU cost, so it should be avoided during gameplay.

Tutorials

- Prototyping levels with CSG

Properties

float	depth	`1.0`
Material	material	

Mode	mode	`0`
bool	path_continuous_u	
float	path_interval	
PathIntervalType	path_interval_type	
bool	path_joined	
bool	path_local	
NodePath	path_node	
PathRotation	path_rotation	
float	path_simplify_angle	
float	path_u_distance	
PackedVector2Array	polygon	`PackedVector2Array(0, 0, 0, 1, 1, 1, 1, 0)`
bool	smooth_faces	`false`

float	spin_degrees
int	spin_sides

Enumerations

enum **Mode**: 🔗

Mode **MODE_DEPTH** = `0`

The polygon shape is extruded along the negative Z axis.

Mode **MODE_SPIN** = `1`

The polygon shape is extruded by rotating it around the Y axis.

Mode **MODE_PATH** = `2`

The polygon shape is extruded along the Path3D specified in path_node.

---

enum **PathRotation**: 🔗

PathRotation **PATH_ROTATION_POLYGON** = `0`

The polygon shape is not rotated.

Note: Requires the path Z coordinates to continually decrease to ensure viable shapes.

PathRotation **PATH_ROTATION_PATH** = `1`

The polygon shape is rotated along the path, but it is not rotated around the path axis.

Note: Requires the path Z coordinates to continually decrease to ensure viable shapes.

PathRotation **PATH_ROTATION_PATH_FOLLOW** = `2`

The polygon shape follows the path and its rotations around the path axis.

---

enum **PathIntervalType**:

PathIntervalType **PATH_INTERVAL_DISTANCE** = `0`

When mode is set to MODE_PATH, path_interval will determine the distance, in meters, each interval of the path will extrude.

PathIntervalType **PATH_INTERVAL_SUBDIVIDE** = `1`

When mode is set to MODE_PATH, path_interval will subdivide the polygons along the path.

Property Descriptions

float **depth** = `1.0`

- void **set_depth**(value: float)
- float **get_depth**()

When mode is MODE_DEPTH, the depth of the extrusion.

Material **material**

- void **set_material**(value: Material)
- Material **get_material**()

Material to use for the resulting mesh. The UV maps the top half of the material to the extruded shape (U along the length of the extrusions and V around the outline of the polygon), the bottom-left quarter to the front end face, and the bottom-right quarter to the back end face.

Mode **mode** = `0` 🔗

- void **set_mode**(value: Mode)
- Mode **get_mode**()

The mode used to extrude the polygon.

---

bool **path_continuous_u** 🔗

- void **set_path_continuous_u**(value: bool)
- bool **is_path_continuous_u**()

When mode is MODE_PATH, by default, the top half of the material is stretched along the entire length of the extruded shape. If `false` the top half of the material is repeated every step of the extrusion.

---

float **path_interval** 🔗

- void **set_path_interval**(value: float)
- float **get_path_interval**()

When mode is MODE_PATH, the path interval or ratio of path points to extrusions.

PathIntervalType **path_interval_type** 🔗

- void **set_path_interval_type**(value: PathIntervalType)
- PathIntervalType **get_path_interval_type**()

When mode is MODE_PATH, this will determine if the interval should be by distance (PATH_INTERVAL_DISTANCE) or subdivision fractions (PATH_INTERVAL_SUBDIVIDE).

---

bool **path_joined** 🔗

- void **set_path_joined**(value: bool)
- bool **is_path_joined**()

When mode is MODE_PATH, if `true` the ends of the path are joined, by adding an extrusion between the last and first points of the path.

---

bool **path_local** 🔗

- void **set_path_local**(value: bool)
- bool **is_path_local**()

When mode is MODE_PATH, if `true` the Transform3D of the **CSGPolygon3D** is used as the starting point for the extrusions, not the Transform3D of the path_node.

NodePath **path_node** 🔗

- void **set_path_node**(value: NodePath)
- NodePath **get_path_node**()

When mode is MODE_PATH, the location of the Path3D object used to extrude the polygon.

PathRotation **path_rotation** 🔗

- void **set_path_rotation**(value: PathRotation)
- PathRotation **get_path_rotation**()

When mode is MODE_PATH, the path rotation method used to rotate the polygon as it is extruded.

---

float **path_simplify_angle** 🔗

- void **set_path_simplify_angle**(value: float)
- float **get_path_simplify_angle**()

When mode is MODE_PATH, extrusions that are less than this angle, will be merged together to reduce polygon count.

---

float **path_u_distance** 🔗

- void **set_path_u_distance**(value: float)
- float **get_path_u_distance**()

When mode is MODE_PATH, this is the distance along the path, in meters, the texture coordinates will tile. When set to 0, texture coordinates will match geometry exactly with no tiling.

PackedVector2Array **polygon** = `PackedVector2Array(0, 0, 0, 1, 1, 1, 1, 0)` 🔗

- void **set_polygon**(value: PackedVector2Array)
- PackedVector2Array **get_polygon**()

The point array that defines the 2D polygon that is extruded. This can be a convex or concave polygon with 3 or more points. The polygon must *not* have any intersecting edges. Otherwise, triangulation will fail and no mesh will be generated.

Note: If only 1 or 2 points are defined in polygon, no mesh will be generated.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

---

bool **smooth_faces** = `false`

- void **set_smooth_faces**(value: bool)
- bool **get_smooth_faces**()

If `true`, applies smooth shading to the extrusions.

---

float **spin_degrees**

- void **set_spin_degrees**(value: float)
- float **get_spin_degrees**()

When mode is MODE_SPIN, the total number of degrees the polygon is rotated when extruding.

---

int **spin_sides**

- void **set_spin_sides**(value: int)
- int **get_spin_sides**()

When mode is MODE_SPIN, the number of extrusions made.

CSGPrimitive3D

Inherits: CSGShape3D **<** GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

Inherited By: CSGBox3D, CSGCylinder3D, CSGMesh3D, CSGPolygon3D, CSGSphere3D, CSGTorus3D

Base class for CSG primitives.

Description

Parent class for various CSG primitives. It contains code and functionality that is common between them. It cannot be used directly. Instead use one of the various classes that inherit from it.

Note: CSG nodes are intended to be used for level prototyping. Creating CSG nodes has a significant CPU cost compared to creating a MeshInstance3D with a PrimitiveMesh. Moving a CSG node within another CSG node also has a significant CPU cost, so it should be avoided during gameplay.

Tutorials

- Prototyping levels with CSG

Properties

bool	flip_faces	`false`

Property Descriptions

bool **flip_faces** = `false` 🔗

- void **set_flip_faces**(value: bool)
- bool **get_flip_faces**()

If set, the order of the vertices in each triangle are reversed resulting in the backside of the mesh being drawn.

CSGShape3D

Inherits: GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

Inherited By: CSGCombiner3D, CSGPrimitive3D

The CSG base class.

Description

This is the CSG base class that provides CSG operation support to the various CSG nodes in Godot.

Note: CSG nodes are intended to be used for level prototyping. Creating CSG nodes has a significant CPU cost compared to creating a MeshInstance3D with a PrimitiveMesh. Moving a CSG node within another CSG node also has a significant CPU cost, so it should be avoided during gameplay.

Tutorials

- Prototyping levels with CSG

Properties

bool	calculate_tangents	`true`
int	collision_layer	`1`

int	collision_mask	`1`
float	collision_priority	`1.0`
Operation	operation	`0`
float	snap	`0.001`
bool	use_collision	`false`

Methods

bool	get_collision_layer_value(layer_number: int) const
bool	get_collision_mask_value(layer_number: int) const
Array	get_meshes() const
bool	is_root_shape() const
void	set_collision_layer_value(layer_number: int, value: bool)
void	set_collision_mask_value(layer_number: int, value: bool)

Enumerations

enum **Operation**:

Operation **OPERATION_UNION** = `0`

Geometry of both primitives is merged, intersecting geometry is removed.

Operation **OPERATION_INTERSECTION** = `1`

Only intersecting geometry remains, the rest is removed.

Operation **OPERATION_SUBTRACTION** = `2`

The second shape is subtracted from the first, leaving a dent with its shape.

Property Descriptions

bool **calculate_tangents** = `true`

- void **set_calculate_tangents**(value: bool)
- bool **is_calculating_tangents**()

Calculate tangents for the CSG shape which allows the use of normal maps. This is only applied on the root shape, this setting is ignored on any child.

int **collision_layer** = `1`

- void **set_collision_layer**(value: int)
- int **get_collision_layer**()

The physics layers this area is in.

Collidable objects can exist in any of 32 different layers. These layers work like a tagging system, and are not visual. A collidable can use these layers to select with which objects it can collide, using the collision_mask property.

A contact is detected if object A is in any of the layers that object B scans, or object B is in any layer scanned by object A. See Collision layers and masks in the documentation for more information.

---

int **collision_mask** = `1` 🔗

- void **set_collision_mask**(value: int)
- int **get_collision_mask**()

The physics layers this CSG shape scans for collisions. Only effective if use_collision is `true`. See Collision layers and masks in the documentation for more information.

---

float **collision_priority** = `1.0` 🔗

- void **set_collision_priority**(value: float)
- float **get_collision_priority**()

The priority used to solve colliding when occurring penetration. Only effective if use_collision is `true`. The higher the priority is, the lower the penetration into the object will be. This can for example be used to prevent the player from breaking through the boundaries of a level.

Operation **operation** = `0` 🔗

- void **set_operation**(value: Operation)

- Operation **get_operation**()

The operation that is performed on this shape. This is ignored for the first CSG child node as the operation is between this node and the previous child of this nodes parent.

---

float **snap** = `0.001`

- void **set_snap**(value: float)
- float **get_snap**()

Snap makes the mesh vertices snap to a given distance so that the faces of two meshes can be perfectly aligned. A lower value results in greater precision but may be harder to adjust. The top-level CSG shape's snap value is used for the entire CSG tree.

---

bool **use_collision** = `false`

- void **set_use_collision**(value: bool)
- bool **is_using_collision**()

Adds a collision shape to the physics engine for our CSG shape. This will always act like a static body. Note that the collision shape is still active even if the CSG shape itself is hidden. See also collision_mask and collision_priority.

Method Descriptions

bool **get_collision_layer_value**(layer_number: int) const

Returns whether or not the specified layer of the collision_layer is enabled, given a `layer_number` between 1 and 32.

---

bool **get_collision_mask_value**(layer_number: int) const 🔗

Returns whether or not the specified layer of the collision_mask is enabled, given a `layer_number` between 1 and 32.

Array **get_meshes**() const 🔗

Returns an Array with two elements, the first is the Transform3D of this node and the second is the root Mesh of this node. Only works when this node is the root shape.

bool **is_root_shape**() const 🔗

Returns `true` if this is a root shape and is thus the object that is rendered.

void **set_collision_layer_value**(layer_number: int, value: bool) 🔗

Based on `value`, enables or disables the specified layer in the collision_layer, given a `layer_number` between 1 and 32.

void **set_collision_mask_value**(layer_number: int, value: bool) 🔗

Based on `value`, enables or disables the specified layer in the collision_mask, given a `layer_number` between 1 and 32.

CSGSphere3D

Inherits: CSGPrimitive3D **<** CSGShape3D **<** GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A CSG Sphere shape.

Description

This node allows you to create a sphere for use with the CSG system.

Note: CSG nodes are intended to be used for level prototyping. Creating CSG nodes has a significant CPU cost compared to creating a MeshInstance3D with a PrimitiveMesh. Moving a CSG node within another CSG node also has a significant CPU cost, so it should be avoided during gameplay.

Tutorials

- Prototyping levels with CSG

Properties

Material	material	
int	radial_segments	`12`

float	radius	`0.5`
int	rings	`6`
bool	smooth_faces	`true`

Property Descriptions

Material **material** 🔗

- void **set_material**(value: Material)
- Material **get_material**()

The material used to render the sphere.

---

int **radial_segments** = `12` 🔗

- void **set_radial_segments**(value: int)
- int **get_radial_segments**()

Number of vertical slices for the sphere.

---

float **radius** = `0.5` 🔗

- void **set_radius**(value: float)
- float **get_radius**()

Radius of the sphere.

---

int **rings** = `6` 🔗

- void **set_rings**(value: int)
- int **get_rings**()

Number of horizontal slices for the sphere.

bool **smooth_faces** = `true` 🔗

- void **set_smooth_faces**(value: bool)
- bool **get_smooth_faces**()

If `true` the normals of the sphere are set to give a smooth effect making the sphere seem rounded. If `false` the sphere will have a flat shaded look.

CSGTorus3D

Inherits: CSGPrimitive3D **<** CSGShape3D **<** GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A CSG Torus shape.

Description

This node allows you to create a torus for use with the CSG system.

Note: CSG nodes are intended to be used for level prototyping. Creating CSG nodes has a significant CPU cost compared to creating a MeshInstance3D with a PrimitiveMesh. Moving a CSG node within another CSG node also has a significant CPU cost, so it should be avoided during gameplay.

Tutorials

- Prototyping levels with CSG

Properties

float	inner_radius	`0.5`
Material	material	
float	outer_radius	`1.0`

int	ring_sides	`6`
int	sides	`8`
bool	smooth_faces	`true`

Property Descriptions

float **inner_radius** = `0.5`

- void **set_inner_radius**(value: float)
- float **get_inner_radius**()

The inner radius of the torus.

Material **material**

- void **set_material**(value: Material)
- Material **get_material**()

The material used to render the torus.

float **outer_radius** = `1.0`

- void **set_outer_radius**(value: float)
- float **get_outer_radius**()

The outer radius of the torus.

int **ring_sides** = `6`

- void **set_ring_sides**(value: int)
- int **get_ring_sides**()

The number of edges each ring of the torus is constructed of.

---

int **sides** = `8`

- void **set_sides**(value: int)
- int **get_sides**()

The number of slices the torus is constructed of.

---

bool **smooth_faces** = `true`

- void **set_smooth_faces**(value: bool)
- bool **get_smooth_faces**()

If `true` the normals of the torus are set to give a smooth effect making the torus seem rounded. If `false` the torus will have a flat shaded look.

DampedSpringJoint2D

Inherits: Joint2D **<** Node2D **<** CanvasItem **<** Node **<** Object

A physics joint that connects two 2D physics bodies with a spring-like force.

Description

A physics joint that connects two 2D physics bodies with a spring-like force. This resembles a spring that always wants to stretch to a given length.

Properties

float	damping	`1.0`
float	length	`50.0`
float	rest_length	`0.0`
float	stiffness	`20.0`

Property Descriptions

float **damping** = `1.0` 🔗

- void **set_damping**(value: float)
- float **get_damping**()

The spring joint's damping ratio. A value between `0` and `1`. When the two bodies move into different directions the system tries to align them to the spring axis again. A high damping value forces the attached bodies to align faster.

---

float **length** = `50.0` 🔗

- void **set_length**(value: float)
- float **get_length**()

The spring joint's maximum length. The two attached bodies cannot stretch it past this value.

---

float **rest_length** = `0.0` 🔗

- void **set_rest_length**(value: float)
- float **get_rest_length**()

When the bodies attached to the spring joint move they stretch or squash it. The joint always tries to resize towards this length.

---

float **stiffness** = `20.0` 🔗

- void **set_stiffness**(value: float)
- float **get_stiffness**()

The higher the value, the less the bodies attached to the joint will deform it. The joint applies an opposing force to the bodies, the product of the stiffness multiplied by the size difference from its resting length.

Decal

Inherits: VisualInstance3D **<** Node3D **<** Node **<** Object

Node that projects a texture onto a MeshInstance3D.

Description

Decals are used to project a texture onto a Mesh in the scene. Use Decals to add detail to a scene without affecting the underlying Mesh. They are often used to add weathering to building, add dirt or mud to the ground, or add variety to props. Decals can be moved at any time, making them suitable for things like blob shadows or laser sight dots.

They are made of an AABB and a group of Texture2Ds specifying Color, normal, ORM (ambient occlusion, roughness, metallic), and emission. Decals are projected within their AABB so altering the orientation of the Decal affects the direction in which they are projected. By default, Decals are projected down (i.e. from positive Y to negative Y).

The Texture2Ds associated with the Decal are automatically stored in a texture atlas which is used for drawing the decals so all decals can be drawn at once. Godot uses clustered decals, meaning they are stored in cluster data and drawn when the mesh is drawn, they are not drawn as a post-processing effect after.

Note: Decals cannot affect an underlying material's transparency, regardless of its transparency mode (alpha blend, alpha scissor, alpha hash, opaque pre-pass). This means translucent or transparent areas of a material will remain translucent or transparent even if an opaque decal is applied on them.

Note: Decals are only supported in the Forward+ and Mobile rendering methods, not Compatibility. When using the Mobile rendering method, only 8 decals can be displayed on each mesh resource. Attempting to display more than 8 decals on a single mesh resource will result in decals flickering in and out as the camera moves.

Note: When using the Mobile rendering method, decals will only correctly affect meshes whose visibility AABB intersects with the decal's AABB. If using a shader to deform the mesh in a way that makes it go outside its AABB, GeometryInstance3D.extra_cull_margin must be increased on the mesh. Otherwise, the decal may not be visible on the mesh.

Properties

float	albedo_mix	`1.0`
int	cull_mask	`1048575`
float	distance_fade_begin	`40.0`
bool	distance_fade_enabled	`false`
float	distance_fade_length	`10.0`
float	emission_energy	`1.0`
float	lower_fade	`0.3`

Color	modulate	`Color(1, 1, 1, 1)`
float	normal_fade	`0.0`
Vector3	size	`Vector3(2, 2, 2)`
Texture2D	texture_albedo	
Texture2D	texture_emission	
Texture2D	texture_normal	
Texture2D	texture_orm	
float	upper_fade	`0.3`

Methods

Texture2D	get_texture(type: DecalTexture) const
void	set_texture(type: DecalTexture, texture: Texture2D)

Enumerations

enum **DecalTexture**: 🔗

DecalTexture **TEXTURE_ALBEDO** = `0`

Texture2D corresponding to texture_albedo.

DecalTexture **TEXTURE_NORMAL** = `1`

Texture2D corresponding to texture_normal.

DecalTexture **TEXTURE_ORM** = `2`

Texture2D corresponding to texture_orm.

DecalTexture **TEXTURE_EMISSION** = `3`

Texture2D corresponding to texture_emission.

DecalTexture **TEXTURE_MAX** = `4`

Max size of DecalTexture enum.

Property Descriptions

float **albedo_mix** = `1.0` 🔗

- void **set_albedo_mix**(value: float)
- float **get_albedo_mix**()

Blends the albedo Color of the decal with albedo Color of the underlying mesh. This can be set to `0.0` to create a decal that only affects normal or ORM. In this case, an albedo texture is still required as its alpha channel will determine where the normal and ORM will be overridden. See also modulate.

---

int **cull_mask** = `1048575` 🔗

- void **set_cull_mask**(value: int)
- int **get_cull_mask**()

Specifies which VisualInstance3D.layers this decal will project on. By default, Decals affect all layers. This is used so you can specify which types of objects receive the Decal and which do not. This is especially useful so you can ensure that dynamic objects don't accidentally receive a Decal intended for the terrain under them.

---

float **distance_fade_begin** = `40.0` 🔗

- void **set_distance_fade_begin**(value: float)
- float **get_distance_fade_begin**()

The distance from the camera at which the Decal begins to fade away (in 3D units).

---

bool **distance_fade_enabled** = `false` 🔗

- void **set_enable_distance_fade**(value: bool)
- bool **is_distance_fade_enabled**()

If `true`, decals will smoothly fade away when far from the active Camera3D starting at distance_fade_begin. The Decal will fade out over distance_fade_begin + distance_fade_length, after which it will be culled and not sent to the shader at all. Use this to reduce the number of active Decals in a scene and thus improve performance.

---

float **distance_fade_length** = `10.0` 🔗

- void **set_distance_fade_length**(value: float)
- float **get_distance_fade_length**()

The distance over which the Decal fades (in 3D units). The Decal becomes slowly more transparent over this distance and is completely invisible at the end. Higher values result in a smoother fade-out transition, which is more suited when the camera moves fast.

---

float **emission_energy** = `1.0` 🔗

- void **set_emission_energy**(value: float)
- float **get_emission_energy**()

Energy multiplier for the emission texture. This will make the decal emit light at a higher or lower intensity, independently of the albedo color. See also modulate.

---

float **lower_fade** = `0.3` 🔗

- void **set_lower_fade**(value: float)
- float **get_lower_fade**()

Sets the curve over which the decal will fade as the surface gets further from the center of the AABB. Only positive values are valid (negative values will be clamped to `0.0`). See also upper_fade.

Color **modulate** = `Color(1, 1, 1, 1)` 🔗

- void **set_modulate**(value: Color)
- Color **get_modulate**()

Changes the Color of the Decal by multiplying the albedo and emission colors with this value. The alpha component is only taken into account when multiplying the albedo color, not the emission color. See also emission_energy and albedo_mix to change the emission and albedo intensity independently of each other.

---

float **normal_fade** = `0.0`

- void **set_normal_fade**(value: float)
- float **get_normal_fade**()

Fades the Decal if the angle between the Decal's AABB and the target surface becomes too large. A value of `0` projects the Decal regardless of angle, a value of `1` limits the Decal to surfaces that are nearly perpendicular.

Note: Setting normal_fade to a value greater than `0.0` has a small performance cost due to the added normal angle computations.

Vector3 **size** = `Vector3(2, 2, 2)`

- void **set_size**(value: Vector3)
- Vector3 **get_size**()

Sets the size of the AABB used by the decal. All dimensions must be set to a value greater than zero (they will be clamped to `0.001` if this is not the case). The AABB goes from `-size/2` to `size/2`.

Note: To improve culling efficiency of "hard surface" decals, set their upper_fade and lower_fade to `0.0` and set the Y component of the size as low as possible. This will reduce the decals' AABB size without affecting their appearance.

Texture2D **texture_albedo** 🔗

- void **set_texture**(type: DecalTexture, texture: Texture2D)
- Texture2D **get_texture**(type: DecalTexture) const

Texture2D with the base Color of the Decal. Either this or the texture_emission must be set for the Decal to be visible. Use the alpha channel like a mask to smoothly blend the edges of the decal with the underlying object.

Note: Unlike BaseMaterial3D whose filter mode can be adjusted on a per-material basis, the filter mode for **Decal** textures is set globally with ProjectSettings.rendering/textures/decals/filter.

Texture2D **texture_emission** 🔗

- void **set_texture**(type: DecalTexture, texture: Texture2D)
- Texture2D **get_texture**(type: DecalTexture) const

Texture2D with the emission Color of the Decal. Either this or the texture_albedo must be set for the Decal to be visible. Use the alpha channel like a mask to smoothly blend the edges of the decal with the underlying object.

Note: Unlike BaseMaterial3D whose filter mode can be adjusted on a per-material basis, the filter mode for **Decal** textures is set globally with ProjectSettings.rendering/textures/decals/filter.

Texture2D **texture_normal** 🔗

- void **set_texture**(type: DecalTexture, texture: Texture2D)
- Texture2D **get_texture**(type: DecalTexture) const

Texture2D with the per-pixel normal map for the decal. Use this to add extra detail to decals.

Note: Unlike BaseMaterial3D whose filter mode can be adjusted on a per-material basis, the filter mode for **Decal** textures is set globally with ProjectSettings.rendering/textures/decals/filter.

Note: Setting this texture alone will not result in a visible decal, as texture_albedo must also be set. To create a normal-only decal, load an albedo texture into texture_albedo and set albedo_mix to `0.0`. The albedo texture's alpha channel will be used to determine where the underlying surface's normal map should be overridden (and its intensity).

Texture2D **texture_orm** 🔗

- void **set_texture**(type: DecalTexture, texture: Texture2D)
- Texture2D **get_texture**(type: DecalTexture) const

Texture2D storing ambient occlusion, roughness, and metallic for the decal. Use this to add extra detail to decals.

Note: Unlike BaseMaterial3D whose filter mode can be adjusted on a per-material basis, the filter mode for **Decal** textures is set globally with ProjectSettings.rendering/textures/decals/filter.

Note: Setting this texture alone will not result in a visible decal, as texture_albedo must also be set. To create an ORM-only decal, load an albedo texture into texture_albedo and set albedo_mix to `0.0`. The albedo texture's alpha channel will be used to determine where the underlying surface's ORM map should be overridden (and its intensity).

---

float **upper_fade** = `0.3` 🔗

- void **set_upper_fade**(value: float)
- float **get_upper_fade**()

Sets the curve over which the decal will fade as the surface gets further from the center of the AABB. Only positive values are valid (negative values will be clamped to `0.0`). See also lower_fade.

Method Descriptions

Texture2D **get_texture**(type: DecalTexture) const 🔗

Returns the Texture2D associated with the specified DecalTexture. This is a convenience method, in most cases you should access the texture directly.

For example, instead of `albedo_tex = $Decal.get_texture(Decal.TEXTURE_ALBEDO)`, use `albedo_tex = $Decal.texture_albedo`.

One case where this is better than accessing the texture directly is when you want to copy one Decal's textures to another. For example:

GDScript

```
for i in Decal.TEXTURE_MAX:
    $NewDecal.set_texture(i, $OldDecal.get_texture(i))
```

C#

```
for (int i = 0; i < (int)Decal.DecalTexture.Max; i++)
{
    GetNode<Decal>("NewDecal").SetTexture(i, GetNode<Decal>
("OldDecal").GetTexture(i));
}
```

---

void **set_texture**(type: DecalTexture, texture: Texture2D) 🔗

Sets the Texture2D associated with the specified DecalTexture. This is a convenience method, in most cases you should access the texture directly.

For example, instead of `$Decal.set_texture(Decal.TEXTURE_ALBEDO, albedo_tex)`, use `$Decal.texture_albedo = albedo_tex`.

One case where this is better than accessing the texture directly is when you want to copy one Decal's textures to another. For example:

GDScript

```
for i in Decal.TEXTURE_MAX:
    $NewDecal.set_texture(i, $OldDecal.get_texture(i))
```

C#

```
for (int i = 0; i < (int)Decal.DecalTexture.Max; i++)
{
    GetNode<Decal>("NewDecal").SetTexture(i, GetNode<Decal>
("OldDecal").GetTexture(i));
}
```

DirectionalLight2D

Inherits: Light2D **<** Node2D **<** CanvasItem **<** Node **<** Object

Directional 2D light from a distance.

Description

A directional light is a type of Light2D node that models an infinite number of parallel rays covering the entire scene. It is used for lights with strong intensity that are located far away from the scene (for example: to model sunlight or moonlight).

Note: DirectionalLight2D does not support light cull masks (but it supports shadow cull masks). It will always light up 2D nodes, regardless of the 2D node's CanvasItem.light_mask.

Tutorials

- 2D lights and shadows

Properties

float	height	`0.0`
float	max_distance	`10000.0`

Property Descriptions

float **height** = `0.0` 🔗

- void **set_height**(value: float)
- float **get_height**()

The height of the light. Used with 2D normal mapping. Ranges from 0 (parallel to the plane) to 1 (perpendicular to the plane).

---

float **max_distance** = `10000.0` 🔗

- void **set_max_distance**(value: float)
- float **get_max_distance**()

The maximum distance from the camera center objects can be before their shadows are culled (in pixels). Decreasing this value can prevent objects located outside the camera from casting shadows (while also improving performance). Camera2D.zoom is not taken into account by max_distance, which means that at higher zoom values, shadows will appear to fade out sooner when zooming onto a given point.

DirectionalLight3D

Inherits: Light3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

Directional light from a distance, as from the Sun.

Description

A directional light is a type of Light3D node that models an infinite number of parallel rays covering the entire scene. It is used for lights with strong intensity that are located far away from the scene to model sunlight or moonlight. The worldspace location of the DirectionalLight3D transform (origin) is ignored. Only the basis is used to determine light direction.

Tutorials

- 3D lights and shadows
- Faking global illumination

Properties

bool	directional_shadow_blend_splits	`false`
float	directional_shadow_fade_start	`0.8`
float	directional_shadow_max_distance	`100.0`

ShadowMode	directional_shadow_mode	`2`
float	directional_shadow_pancake_size	`20.0`
float	directional_shadow_split_1	`0.1`
float	directional_shadow_split_2	`0.2`
float	directional_shadow_split_3	`0.5`
SkyMode	sky_mode	`0`

Enumerations

enum **ShadowMode**: 🔗

ShadowMode **SHADOW_ORTHOGONAL** = `0`

Renders the entire scene's shadow map from an orthogonal point of view. This is the fastest directional shadow mode. May result in blurrier shadows on close objects.

ShadowMode **SHADOW_PARALLEL_2_SPLITS** = `1`

Splits the view frustum in 2 areas, each with its own shadow map. This shadow mode is a compromise between SHADOW_ORTHOGONAL and SHADOW_PARALLEL_4_SPLITS in terms of performance.

ShadowMode **SHADOW_PARALLEL_4_SPLITS** = `2`

Splits the view frustum in 4 areas, each with its own shadow map. This is the slowest directional shadow mode.

---

enum **SkyMode**: 🔗

SkyMode **SKY_MODE_LIGHT_AND_SKY** = `0`

Makes the light visible in both scene lighting and sky rendering.

SkyMode **SKY_MODE_LIGHT_ONLY** = `1`

Makes the light visible in scene lighting only (including direct lighting and global illumination). When using this mode, the light will not be visible from sky shaders.

SkyMode **SKY_MODE_SKY_ONLY** = `2`

Makes the light visible to sky shaders only. When using this mode the light will not cast light into the scene (either through direct lighting or through global illumination), but can be accessed through sky shaders. This can be useful, for example, when you want to control sky effects without illuminating the scene (during a night cycle, for example).

Property Descriptions

bool **directional_shadow_blend_splits** = `false` 🔗

- void **set_blend_splits**(value: bool)
- bool **is_blend_splits_enabled**()

If `true`, shadow detail is sacrificed in exchange for smoother transitions between splits. Enabling shadow blend splitting also has a moderate performance cost. This is ignored when directional_shadow_mode is SHADOW_ORTHOGONAL.

---

float **directional_shadow_fade_start** = `0.8`

- void **set_param**(value: float)
- float **get_param**()

Proportion of directional_shadow_max_distance at which point the shadow starts to fade. At directional_shadow_max_distance, the shadow will disappear. The default value is a balance between smooth fading and distant shadow visibility. If the camera moves fast and the directional_shadow_max_distance is low, consider lowering directional_shadow_fade_start below `0.8` to make shadow transitions less noticeable. On the other hand, if you tuned directional_shadow_max_distance to cover the entire scene, you can set directional_shadow_fade_start to `1.0` to prevent the shadow from fading in the distance (it will suddenly cut off instead).

---

float **directional_shadow_max_distance** = `100.0`

- void **set_param**(value: float)
- float **get_param**()

The maximum distance for shadow splits. Increasing this value will make directional shadows visible from further away, at the cost of lower overall shadow detail and performance (since more objects need to be included in the directional shadow rendering).

ShadowMode **directional_shadow_mode** = `2`

- void **set_shadow_mode**(value: ShadowMode)
- ShadowMode **get_shadow_mode**()

The light's shadow rendering algorithm. See ShadowMode.

---

float **directional_shadow_pancake_size** = `20.0` 🔗

- void **set_param**(value: float)
- float **get_param**()

Sets the size of the directional shadow pancake. The pancake offsets the start of the shadow's camera frustum to provide a higher effective depth resolution for the shadow. However, a high pancake size can cause artifacts in the shadows of large objects that are close to the edge of the frustum. Reducing the pancake size can help. Setting the size to `0` turns off the pancaking effect.

---

float **directional_shadow_split_1** = `0.1` 🔗

- void **set_param**(value: float)
- float **get_param**()

The distance from camera to shadow split 1. Relative to directional_shadow_max_distance. Only used when directional_shadow_mode is SHADOW_PARALLEL_2_SPLITS or SHADOW_PARALLEL_4_SPLITS.

---

float **directional_shadow_split_2** = `0.2` 🔗

- void **set_param**(value: float)
- float **get_param**()

The distance from shadow split 1 to split 2. Relative to directional_shadow_max_distance. Only used when directional_shadow_mode is SHADOW_PARALLEL_4_SPLITS.

---

float **directional_shadow_split_3** = `0.5`

- void **set_param**(value: float)
- float **get_param**()

The distance from shadow split 2 to split 3. Relative to directional_shadow_max_distance. Only used when directional_shadow_mode is SHADOW_PARALLEL_4_SPLITS.

SkyMode **sky_mode** = `0`

- void **set_sky_mode**(value: SkyMode)
- SkyMode **get_sky_mode**()

Set whether this **DirectionalLight3D** is visible in the sky, in the scene, or both in the sky and in the scene. See SkyMode for options.

EditorCommandPalette

Inherits: ConfirmationDialog **<** AcceptDialog **<** Window **<** Viewport **<** Node **<** Object

Godot editor's command palette.

Description

Object that holds all the available Commands and their shortcuts text. These Commands can be accessed through **Editor > Command Palette** menu.

Command key names use slash delimiters to distinguish sections, for example: `"example/command1"` then `example` will be the section name.

GDScript

```
var command_palette = EditorInterface.get_command_palette()
# external_command is a function that will be called with the
command is executed.
var command_callable = Callable(self,
"external_command").bind(arguments)
command_palette.add_command("command",
"test/command",command_callable)
```

C#

```
EditorCommandPalette commandPalette =
EditorInterface.Singleton.GetCommandPalette();
// ExternalCommand is a function that will be called with the
command is executed.
Callable commandCallable = new Callable(this,
MethodName.ExternalCommand);
```

```
commandPalette.AddCommand("command", "test/command",
commandCallable)
```

Note: This class shouldn't be instantiated directly. Instead, access the singleton using EditorInterface.get_command_palette.

Properties

bool	dialog_hide_on_ok	`false`	(overrides AcceptDialog)

Methods

void	add_command(command_name: String, key_name: String, binded_callable: Callable, shortcut_text: String = "None")
void	remove_command(key_name: String)

Method Descriptions

void **add_command**(command_name: String, key_name: String, binded_callable: Callable, shortcut_text: String = "None") 🔗

Adds a custom command to EditorCommandPalette.

- `command_name`: String (Name of the **Command**. This is displayed to the user.)
- `key_name`: String (Name of the key for a particular **Command**. This is used to uniquely identify the **Command**.)

- `binded_callable`: Callable (Callable of the **Command**. This will be executed when the **Command** is selected.)
- `shortcut_text`: String (Shortcut text of the **Command** if available.)

---

void **remove_command**(key_name: String) 🔗

Removes the custom command from EditorCommandPalette.

- `key_name`: String (Name of the key for a particular **Command**.)

EditorFileDialog

Inherits: ConfirmationDialog **<** AcceptDialog **<** Window **<** Viewport **<** Node **<** Object

A modified version of FileDialog used by the editor.

Description

EditorFileDialog is an enhanced version of FileDialog available only to editor plugins. Additional features include list of favorited/recent files and the ability to see files as thumbnails grid instead of list.

Properties

Access	access	`0`
String	current_dir	
String	current_file	
String	current_path	
bool	dialog_hide_on_ok	`false` (overrides AcceptDialog)
bool	disable_overwrite_warning	`false`

DisplayMode	display_mode	`0`
FileMode	file_mode	`4`
PackedStringArray	filters	`PackedStringArray()`
int	option_count	`0`
bool	show_hidden_files	`false`
String	title	`"Save a File"` (overrides Window)

Methods

void	add_filter(filter: String, description: String = "")
void	add_option(name: String, values: PackedStringArray, default_value_index: int)
void	add_side_menu(menu: Control, title: String = "")
void	clear_filters()

LineEdit	get_line_edit()
int	get_option_default(option: int) const
String	get_option_name(option: int) const
PackedStringArray	get_option_values(option: int) const
Dictionary	get_selected_options() const
VBoxContainer	get_vbox()
void	invalidate()
void	popup_file_dialog()
void	set_option_default(option: int, default_value_index: int)
void	set_option_name(option: int, name: String)
void	set_option_values(option: int, values: PackedStringArray)

Signals

dir_selected(dir: String) 🔗

Emitted when a directory is selected.

---

file_selected(path: String) 🔗

Emitted when a file is selected.

---

files_selected(paths: PackedStringArray) 🔗

Emitted when multiple files are selected.

Enumerations

enum **FileMode**: 🔗

FileMode **FILE_MODE_OPEN_FILE** = `0`

The **EditorFileDialog** can select only one file. Accepting the window will open the file.

FileMode **FILE_MODE_OPEN_FILES** = `1`

The **EditorFileDialog** can select multiple files. Accepting the window will open all files.

FileMode **FILE_MODE_OPEN_DIR** = `2`

The **EditorFileDialog** can select only one directory. Accepting the window will open the directory.

FileMode **FILE_MODE_OPEN_ANY** = `3`

The **EditorFileDialog** can select a file or directory. Accepting the window will open it.

FileMode **FILE_MODE_SAVE_FILE** = `4`

The **EditorFileDialog** can select only one file. Accepting the window will save the file.

---

enum **Access**:

Access **ACCESS_RESOURCES** = `0`

The **EditorFileDialog** can only view `res://` directory contents.

Access **ACCESS_USERDATA** = `1`

The **EditorFileDialog** can only view `user://` directory contents.

Access **ACCESS_FILESYSTEM** = `2`

The **EditorFileDialog** can view the entire local file system.

---

enum **DisplayMode**:

DisplayMode **DISPLAY_THUMBNAILS** = `0`

The **EditorFileDialog** displays resources as thumbnails.

DisplayMode **DISPLAY_LIST** = `1`

The **EditorFileDialog** displays resources as a list of filenames.

Property Descriptions

Access **access** = `0`

- void **set_access**(value: Access)
- Access **get_access**()

The location from which the user may select a file, including `res://`, `user://`, and the local file system.

String **current_dir**

- void **set_current_dir**(value: String)
- String **get_current_dir**()

The currently occupied directory.

String **current_file**

- void **set_current_file**(value: String)
- String **get_current_file**()

The currently selected file.

String **current_path**

- void **set_current_path**(value: String)
- String **get_current_path**()

The file system path in the address bar.

---

bool **disable_overwrite_warning** = `false`

- void **set_disable_overwrite_warning**(value: bool)
- bool **is_overwrite_warning_disabled**()

If `true`, the **EditorFileDialog** will not warn the user before overwriting files.

DisplayMode **display_mode** = `0` 🔗

- void **set_display_mode**(value: DisplayMode)
- DisplayMode **get_display_mode**()

The view format in which the **EditorFileDialog** displays resources to the user.

FileMode **file_mode** = `4` 🔗

- void **set_file_mode**(value: FileMode)
- FileMode **get_file_mode**()

The dialog's open or save mode, which affects the selection behavior. See FileMode.

PackedStringArray **filters** = `PackedStringArray()` 🔗

- void **set_filters**(value: PackedStringArray)
- PackedStringArray **get_filters**()

The available file type filters. For example, this shows only `.png` and `.gd` files: `set_filters(PackedStringArray(["*.png ; PNG Images","*.gd ; GDScript Files"]))`. Multiple file types can also be specified in a single filter. `"*.png, *.jpg, *.jpeg ; Supported Images"` will show both PNG and JPEG files when selected.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedStringArray for more details.

---

int **option_count** = `0` 🔗

- void **set_option_count**(value: int)
- int **get_option_count**()

The number of additional OptionButtons and CheckBoxes in the dialog.

---

bool **show_hidden_files** = `false` 🔗

- void **set_show_hidden_files**(value: bool)
- bool **is_showing_hidden_files**()

If `true`, hidden files and directories will be visible in the **EditorFileDialog**. This property is synchronized with EditorSettings.filesystem/file_dialog/show_hidden_files.

Method Descriptions

void **add_filter**(filter: String, description: String = "") 🔗

Adds a comma-delimited file name `filter` option to the **EditorFileDialog** with an optional `description`, which restricts what files can be picked.

A `filter` should be of the form `"filename.extension"`, where filename and extension can be `*` to match any string. Filters starting with `.` (i.e. empty filenames) are not allowed.

For example, a `filter` of `"*.tscn, *.scn"` and a `description` of `"Scenes"` results in filter text "Scenes (*.tscn, *.scn)".

---

void **add_option**(name: String, values: PackedStringArray, default_value_index: int) 🔗

Adds an additional OptionButton to the file dialog. If `values` is empty, a CheckBox is added instead.

`default_value_index` should be an index of the value in the `values`. If `values` is empty it should be either `1` (checked), or `0` (unchecked).

---

void **add_side_menu**(menu: Control, title: String = "") 🔗

Adds the given `menu` to the side of the file dialog with the given `title` text on top. Only one side menu is allowed.

---

void **clear_filters**() 🔗

Removes all filters except for "All Files (*)".

LineEdit **get_line_edit**() 🔗

Returns the LineEdit for the selected file.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their CanvasItem.visible property.

---

int **get_option_default**(option: int) const 🔗

Returns the default value index of the OptionButton or CheckBox with index `option`.

String **get_option_name**(option: int) const

Returns the name of the OptionButton or CheckBox with index `option`.

PackedStringArray **get_option_values**(option: int) const

Returns an array of values of the OptionButton with index `option`.

Dictionary **get_selected_options**() const

Returns a Dictionary with the selected values of the additional OptionButtons and/or CheckBoxes. Dictionary keys are names and values are selected value indices.

VBoxContainer **get_vbox**()

Returns the VBoxContainer used to display the file system.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their CanvasItem.visible property.

---

void **invalidate**()

Notify the **EditorFileDialog** that its view of the data is no longer accurate. Updates the view contents on next view update.

---

void **popup_file_dialog**() 🔗

Shows the **EditorFileDialog** at the default size and position for file dialogs in the editor, and selects the file name if there is a current file.

---

void **set_option_default**(option: int, default_value_index: int) 🔗

Sets the default value index of the OptionButton or CheckBox with index `option`.

---

void **set_option_name**(option: int, name: String) 🔗

Sets the name of the OptionButton or CheckBox with index `option`.

---

void **set_option_values**(option: int, values: PackedStringArray) 🔗

Sets the option values of the OptionButton with index `option`.

EditorFileSystem

Inherits: Node **<** Object

Resource filesystem, as the editor sees it.

Description

This object holds information of all resources in the filesystem, their types, etc.

Note: This class shouldn't be instantiated directly. Instead, access the singleton using EditorInterface.get_resource_filesystem.

Methods

String	get_file_type(path: String) const
EditorFileSystemDirectory	get_filesystem()
EditorFileSystemDirectory	get_filesystem_path(path: String)
float	get_scanning_progress() const
bool	is_scanning() const
void	reimport_files(files: PackedStringArray)

void	scan()
void	scan_sources()
void	update_file(path: String)

Signals

filesystem_changed()

Emitted if the filesystem changed.

resources_reimported(resources: PackedStringArray)

Emitted if a resource is reimported.

resources_reimporting(resources: PackedStringArray)

Emitted before a resource is reimported.

resources_reload(resources: PackedStringArray)

Emitted if at least one resource is reloaded when the filesystem is scanned.

script_classes_updated()

Emitted when the list of global script classes gets updated.

sources_changed(exist: bool)

Emitted if the source of any imported file changed.

Method Descriptions

String **get_file_type**(path: String) const

Returns the resource type of the file, given the full path. This returns a string such as `"Resource"` or `"GDScript"`, *not* a file extension such as `".gd"`.

EditorFileSystemDirectory **get_filesystem**()

Gets the root directory object.

EditorFileSystemDirectory **get_filesystem_path**(path: String)

Returns a view into the filesystem at `path`.

float **get_scanning_progress**() const

Returns the scan progress for 0 to 1 if the FS is being scanned.

bool **is_scanning**() const

Returns `true` if the filesystem is being scanned.

void **reimport_files**(files: PackedStringArray)

Reimports a set of files. Call this if these files or their `.import` files were directly edited by script or an external program.

If the file type changed or the file was newly created, use update_file or scan.

Note: This function blocks until the import is finished. However, the main loop iteration, including timers and Node._process, will occur during the import process due to progress bar updates. Avoid calls to reimport_files or scan while an import is in progress.

---

void **scan**()

Scan the filesystem for changes.

---

void **scan_sources**()

Check if the source of any imported resource changed.

---

void **update_file**(path: String)

Add a file in an existing directory, or schedule file information to be updated on editor restart. Can be used to update text files saved by an external program.

This will not import the file. To reimport, call reimport_files or scan methods.

EditorInspector

Inherits: ScrollContainer **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

A control used to edit properties of an object.

Description

This is the control that implements property editing in the editor's Settings dialogs, the Inspector dock, etc. To get the **EditorInspector** used in the editor's Inspector dock, use EditorInterface.get_inspector.

EditorInspector will show properties in the same order as the array returned by Object.get_property_list.

If a property's name is path-like (i.e. if it contains forward slashes), **EditorInspector** will create nested sections for "directories" along the path. For example, if a property is named `highlighting/gdscript/node_path_color`, it will be shown as "Node Path Color" inside the "GDScript" section nested inside the "Highlighting" section.

If a property has @GlobalScope.PROPERTY_USAGE_GROUP usage, it will group subsequent properties whose name starts with the property's hint string. The group ends when a property does not start with that hint string or when a new group starts. An empty group name effectively ends the current group. **EditorInspector** will create a top-level section for each group. For example, if a property with group usage is named `Collide With` and its hint string is `collide_with_`, a subsequent `collide_with_area` property will be shown as "Area" inside the "Collide With" section. There is also a special case: when the hint string contains the name of a property, that property is grouped too. This is mainly to help grouping properties like `font`, `font_color` and `font_size` (using the hint string `font_`).

If a property has @GlobalScope.PROPERTY_USAGE_SUBGROUP usage, a subgroup will be created in the same way as a group, and a second-level section will be created for each subgroup.

Note: Unlike sections created from path-like property names, **EditorInspector** won't capitalize the name for sections created from groups. So properties with group usage usually use capitalized names instead of snake_cased names.

Properties

ScrollMode	horizontal_scroll_mode	`0` (overrides ScrollContainer)

Methods

Object	get_edited_object()
String	get_selected_path() const

Signals

edited_object_changed() 🔗

Emitted when the object being edited by the inspector has changed.

object_id_selected(id: int)

Emitted when the Edit button of an Object has been pressed in the inspector. This is mainly used in the remote scene tree Inspector.

property_deleted(property: String)

Emitted when a property is removed from the inspector.

property_edited(property: String)

Emitted when a property is edited in the inspector.

property_keyed(property: String, value: Variant, advance: bool)

Emitted when a property is keyed in the inspector. Properties can be keyed by clicking the "key" icon next to a property when the Animation panel is toggled.

property_selected(property: String)

Emitted when a property is selected in the inspector.

property_toggled(property: String, checked: bool)

Emitted when a boolean property is toggled in the inspector.

Note: This signal is never emitted if the internal `autoclear` property enabled. Since this property is always enabled in the editor inspector, this signal is never emitted by the editor itself.

---

resource_selected(resource: Resource, path: String) 🔗

Emitted when a resource is selected in the inspector.

---

restart_requested() 🔗

Emitted when a property that requires a restart to be applied is edited in the inspector. This is only used in the Project Settings and Editor Settings.

Method Descriptions

Object **get_edited_object**() 🔗

Returns the object currently selected in this inspector.

String **get_selected_path**() const 🔗

Gets the path of the currently selected property.

EditorPlugin

Inherits: Node **<** Object

Used by the editor to extend its functionality.

Description

Plugins are used by the editor to extend functionality. The most common types of plugins are those which edit a given node or resource type, import plugins and export plugins. See also EditorScript to add functions to the editor.

Note: Some names in this class contain "left" or "right" (e.g. DOCK_SLOT_LEFT_UL). These APIs assume left-to-right layout, and would be backwards when using right-to-left layout. These names are kept for compatibility reasons.

Tutorials

- Editor plugins documentation index

Methods

void	_apply_changes() virtual
bool	_build() virtual
void	_clear() virtual
void	_disable_plugin() virtual
void	_edit(object: Object) virtual
void	_enable_plugin() virtual
void	_forward_3d_draw_over_viewport(viewport_control: Control) virtual

void	_forward_3d_force_draw_over_viewport(viewport_control: Control) virtual
int	_forward_3d_gui_input(viewport_camera: Camera3D, event: InputEvent) virtual
void	_forward_canvas_draw_over_viewport(viewport_control: Control) virtual
void	_forward_canvas_force_draw_over_viewport(viewport_control: Control) virtual
bool	_forward_canvas_gui_input(event: InputEvent) virtual
PackedStringArray	_get_breakpoints() virtual const
Texture2D	_get_plugin_icon() virtual const
String	_get_plugin_name() virtual const
Dictionary	_get_state() virtual const
String	_get_unsaved_status(for_scene: String) virtual const
void	_get_window_layout(configuration: ConfigFile) virtual
bool	_handles(object: Object) virtual const
bool	_has_main_screen() virtual const
void	_make_visible(visible: bool) virtual
void	_save_external_data() virtual

void	_set_state(state: Dictionary) virtual
void	_set_window_layout(configuration: ConfigFile) virtual
void	add_autoload_singleton(name: String, path: String)
Button	add_control_to_bottom_panel(control: Control, title: String, shortcut: Shortcut = null)
void	add_control_to_container(container: CustomControlContainer, control: Control)
void	add_control_to_dock(slot: DockSlot, control: Control, shortcut: Shortcut = null)
void	add_custom_type(type: String, base: String, script: Script, icon: Texture2D)
void	add_debugger_plugin(script: EditorDebuggerPlugin)
void	add_export_plugin(plugin: EditorExportPlugin)
void	add_import_plugin(importer: EditorImportPlugin, first_priority: bool = false)
void	add_inspector_plugin(plugin: EditorInspectorPlugin)
void	add_node_3d_gizmo_plugin(plugin: EditorNode3DGizmoPlugin)
void	add_resource_conversion_plugin(plugin: EditorResourceConversionPlugin)
void	add_scene_format_importer_plugin(scene_format_importer: EditorSceneFormatImporter, first_priority: bool = false)

void	add_scene_post_import_plugin(scene_import_plugin: EditorScenePostImportPlugin, first_priority: bool = false)
void	add_tool_menu_item(name: String, callable: Callable)
void	add_tool_submenu_item(name: String, submenu: PopupMenu)
void	add_translation_parser_plugin(parser: EditorTranslationParserPlugin)
void	add_undo_redo_inspector_hook_callback(callable: Callable)
EditorInterface	get_editor_interface()
PopupMenu	get_export_as_menu()
String	get_plugin_version() const
ScriptCreateDialog	get_script_create_dialog()
EditorUndoRedoManager	get_undo_redo()
void	hide_bottom_panel()
void	make_bottom_panel_item_visible(item: Control)
void	queue_save_layout()
void	remove_autoload_singleton(name: String)
void	remove_control_from_bottom_panel(control: Control)
void	remove_control_from_container(container: CustomControlContainer, control: Control)

void	remove_control_from_docks(control: Control)
void	remove_custom_type(type: String)
void	remove_debugger_plugin(script: EditorDebuggerPlugin)
void	remove_export_plugin(plugin: EditorExportPlugin)
void	remove_import_plugin(importer: EditorImportPlugin)
void	remove_inspector_plugin(plugin: EditorInspectorPlugin)
void	remove_node_3d_gizmo_plugin(plugin: EditorNode3DGizmoPlugin)
void	remove_resource_conversion_plugin(plugin: EditorResourceConversionPlugin)
void	remove_scene_format_importer_plugin(scene_format_importer: EditorSceneFormatImporter)
void	remove_scene_post_import_plugin(scene_import_plugin: EditorScenePostImportPlugin)
void	remove_tool_menu_item(name: String)
void	remove_translation_parser_plugin(parser: EditorTranslationParserPlugin)
void	remove_undo_redo_inspector_hook_callback(callable: Callable)
void	set_dock_tab_icon(control: Control, icon: Texture2D)
void	set_force_draw_over_forwarding_enabled()

void	set_input_event_forwarding_always_enabled()
int	update_overlays() const

Signals

main_screen_changed(screen_name: String)

Emitted when user changes the workspace (**2D**, **3D**, **Script**, **AssetLib**). Also works with custom screens defined by plugins.

---

project_settings_changed()

Deprecated: Use ProjectSettings.settings_changed instead.

Emitted when any project setting has changed.

---

resource_saved(resource: Resource)

Emitted when the given `resource` was saved on disc. See also scene_saved.

---

scene_changed(scene_root: Node)

Emitted when the scene is changed in the editor. The argument will return the root node of the scene that has just become active. If this scene is new and empty, the argument will be `null`.

---

scene_closed(filepath: String)

Emitted when user closes a scene. The argument is a file path to the closed scene.

---

scene_saved(filepath: String)

Emitted when a scene was saved on disc. The argument is a file path to the saved scene. See also resource_saved.

Enumerations

enum **CustomControlContainer**: 🔗

CustomControlContainer **CONTAINER_TOOLBAR** = `0`

Main editor toolbar, next to play buttons.

CustomControlContainer **CONTAINER_SPATIAL_EDITOR_MENU** = `1`

The toolbar that appears when 3D editor is active.

CustomControlContainer **CONTAINER_SPATIAL_EDITOR_SIDE_LEFT** = `2`

Left sidebar of the 3D editor.

CustomControlContainer **CONTAINER_SPATIAL_EDITOR_SIDE_RIGHT** = `3`

Right sidebar of the 3D editor.

CustomControlContainer **CONTAINER_SPATIAL_EDITOR_BOTTOM** = `4`

Bottom panel of the 3D editor.

CustomControlContainer **CONTAINER_CANVAS_EDITOR_MENU** = `5`

The toolbar that appears when 2D editor is active.

CustomControlContainer **CONTAINER_CANVAS_EDITOR_SIDE_LEFT** = `6`

Left sidebar of the 2D editor.

CustomControlContainer **CONTAINER_CANVAS_EDITOR_SIDE_RIGHT** = `7`

Right sidebar of the 2D editor.

CustomControlContainer **CONTAINER_CANVAS_EDITOR_BOTTOM** = `8`

Bottom panel of the 2D editor.

CustomControlContainer **CONTAINER_INSPECTOR_BOTTOM** = `9`

Bottom section of the inspector.

CustomControlContainer **CONTAINER_PROJECT_SETTING_TAB_LEFT** = `10`

Tab of Project Settings dialog, to the left of other tabs.

CustomControlContainer **CONTAINER_PROJECT_SETTING_TAB_RIGHT** = `11`

Tab of Project Settings dialog, to the right of other tabs.

---

enum **DockSlot**: 🔗

DockSlot **DOCK_SLOT_LEFT_UL** = `0`

Dock slot, left side, upper-left (empty in default layout).

DockSlot **DOCK_SLOT_LEFT_BL** = `1`

Dock slot, left side, bottom-left (empty in default layout).

DockSlot **DOCK_SLOT_LEFT_UR** = `2`

Dock slot, left side, upper-right (in default layout includes Scene and Import docks).

DockSlot **DOCK_SLOT_LEFT_BR** = `3`

Dock slot, left side, bottom-right (in default layout includes FileSystem dock).

DockSlot **DOCK_SLOT_RIGHT_UL** = `4`

Dock slot, right side, upper-left (in default layout includes Inspector, Node, and History docks).

DockSlot **DOCK_SLOT_RIGHT_BL** = `5`

Dock slot, right side, bottom-left (empty in default layout).

DockSlot **DOCK_SLOT_RIGHT_UR** = `6`

Dock slot, right side, upper-right (empty in default layout).

DockSlot **DOCK_SLOT_RIGHT_BR** = `7`

Dock slot, right side, bottom-right (empty in default layout).

DockSlot **DOCK_SLOT_MAX** = `8`

Represents the size of the DockSlot enum.

---

enum **AfterGUIInput**: 🔗

AfterGUIInput **AFTER_GUI_INPUT_PASS** = `0`

Forwards the InputEvent to other EditorPlugins.

AfterGUIInput **AFTER_GUI_INPUT_STOP** = `1`

Prevents the InputEvent from reaching other Editor classes.

AfterGUIInput **AFTER_GUI_INPUT_CUSTOM** = `2`

Pass the InputEvent to other editor plugins except the main Node3D one. This can be used to prevent node selection changes and work with sub-gizmos instead.

Method Descriptions

void **_apply_changes**() virtual

This method is called when the editor is about to save the project, switch to another tab, etc. It asks the plugin to apply any pending state changes to ensure consistency.

This is used, for example, in shader editors to let the plugin know that it must apply the shader code being written by the user to the object.

---

bool **_build**() virtual

This method is called when the editor is about to run the project. The plugin can then perform required operations before the project runs.

This method must return a boolean. If this method returns `false`, the project will not run. The run is aborted immediately, so this also prevents all other plugins' _build methods from running.

---

void **_clear**() virtual

Clear all the state and reset the object being edited to zero. This ensures your plugin does not keep editing a currently existing node, or a node from the wrong scene.

---

void **_disable_plugin**() virtual

Called by the engine when the user disables the **EditorPlugin** in the Plugin tab of the project settings window.

---

void **_edit**(object: Object) virtual

This function is used for plugins that edit specific object types (nodes or resources). It requests the editor to edit the given object.

`object` can be `null` if the plugin was editing an object, but there is no longer any selected object handled by this plugin. It can be used to cleanup editing state.

---

void **_enable_plugin**() virtual

Called by the engine when the user enables the **EditorPlugin** in the Plugin tab of the project settings window.

---

void **_forward_3d_draw_over_viewport**(viewport_control: Control) virtual

Called by the engine when the 3D editor's viewport is updated. Use the `overlay` Control for drawing. You can update the viewport manually by calling update_overlays.

GDScript

```
func _forward_3d_draw_over_viewport(overlay):
    # Draw a circle at cursor position.
    overlay.draw_circle(overlay.get_local_mouse_position(), 64, Color.WHITE)

func _forward_3d_gui_input(camera, event):
    if event is InputEventMouseMotion:
        # Redraw viewport when cursor is moved.
        update_overlays()
        return EditorPlugin.AFTER_GUI_INPUT_STOP
    return EditorPlugin.AFTER_GUI_INPUT_PASS
```

C#

```
public override void _Forward3DDrawOverViewport(Control viewportControl)
{
    // Draw a circle at cursor position.
    viewportControl.DrawCircle(viewportControl.GetLocalMousePosition(), 64,
Colors.White);
}

public override EditorPlugin.AfterGuiInput _Forward3DGuiInput(Camera3D
viewportCamera, InputEvent @event)
{
    if (@event is InputEventMouseMotion)
    {
        // Redraw viewport when cursor is moved.
        UpdateOverlays();
        return EditorPlugin.AfterGuiInput.Stop;
    }
    return EditorPlugin.AfterGuiInput.Pass;
}
```

---

void **_forward_3d_force_draw_over_viewport**(viewport_control: Control) virtual

This method is the same as _forward_3d_draw_over_viewport, except it draws on top of everything. Useful when you need an extra layer that shows over anything else.

You need to enable calling of this method by using set_force_draw_over_forwarding_enabled.

---

int **_forward_3d_gui_input**(viewport_camera: Camera3D, event: InputEvent) virtual 🔗

Called when there is a root node in the current edited scene, _handles is implemented, and an InputEvent happens in the 3D viewport. The return value decides whether the InputEvent is consumed or forwarded to other **EditorPlugin**s. See AfterGUIInput for options.

Example:

GDScript

```
# Prevents the InputEvent from reaching other Editor classes.
func _forward_3d_gui_input(camera, event):
    return EditorPlugin.AFTER_GUI_INPUT_STOP
```

C#

```
// Prevents the InputEvent from reaching other Editor classes.
public override EditorPlugin.AfterGuiInput _Forward3DGuiInput(Camera3D camera,
InputEvent @event)
{
    return EditorPlugin.AfterGuiInput.Stop;
}
```

Must `return EditorPlugin.AFTER_GUI_INPUT_PASS` in order to forward the InputEvent to other Editor classes.

Example:

GDScript

```
# Consumes InputEventMouseMotion and forwards other InputEvent types.
func _forward_3d_gui_input(camera, event):
    return EditorPlugin.AFTER_GUI_INPUT_STOP if event is InputEventMouseMotion else
EditorPlugin.AFTER_GUI_INPUT_PASS
```

C#

```
// Consumes InputEventMouseMotion and forwards other InputEvent types.
public override EditorPlugin.AfterGuiInput _Forward3DGuiInput(Camera3D camera,
InputEvent @event)
{
    return @event is InputEventMouseMotion ? EditorPlugin.AfterGuiInput.Stop :
EditorPlugin.AfterGuiInput.Pass;
}
```

---

void **_forward_canvas_draw_over_viewport**(viewport_control: Control) virtual 🔗

Called by the engine when the 2D editor's viewport is updated. Use the `overlay` Control for drawing. You can update the viewport manually by calling update_overlays.

GDScript

```
func _forward_canvas_draw_over_viewport(overlay):
    # Draw a circle at cursor position.
    overlay.draw_circle(overlay.get_local_mouse_position(), 64, Color.WHITE)

func _forward_canvas_gui_input(event):
    if event is InputEventMouseMotion:
        # Redraw viewport when cursor is moved.
        update_overlays()
        return true
    return false
```

C#

```
public override void _ForwardCanvasDrawOverViewport(Control viewportControl)
{
    // Draw a circle at cursor position.
    viewportControl.DrawCircle(viewportControl.GetLocalMousePosition(), 64,
Colors.White);
}

public override bool _ForwardCanvasGuiInput(InputEvent @event)
{
    if (@event is InputEventMouseMotion)
    {
        // Redraw viewport when cursor is moved.
        UpdateOverlays();
        return true;
    }
    return false;
}
```

---

void **_forward_canvas_force_draw_over_viewport**(viewport_control: Control) virtual 🔗

This method is the same as _forward_canvas_draw_over_viewport, except it draws on top of everything. Useful when you need an extra layer that shows over anything else.

You need to enable calling of this method by using set_force_draw_over_forwarding_enabled.

---

bool **_forward_canvas_gui_input**(event: InputEvent) virtual 🔗

Called when there is a root node in the current edited scene, _handles is implemented and an InputEvent happens in the 2D viewport. Intercepts the InputEvent, if `return true` **EditorPlugin** consumes the `event`, otherwise forwards `event` to other Editor classes.

Example:

GDScript

```
# Prevents the InputEvent from reaching other Editor classes.
func _forward_canvas_gui_input(event):
    return true
```

C#

```
// Prevents the InputEvent from reaching other Editor classes.
public override bool ForwardCanvasGuiInput(InputEvent @event)
{
    return true;
}
```

Must `return false` in order to forward the InputEvent to other Editor classes.

Example:

GDScript

```
# Consumes InputEventMouseMotion and forwards other InputEvent types.
func _forward_canvas_gui_input(event):
    if (event is InputEventMouseMotion):
        return true
    return false
```

C#

```
// Consumes InputEventMouseMotion and forwards other InputEvent types.
public override bool _ForwardCanvasGuiInput(InputEvent @event)
{
    if (@event is InputEventMouseMotion)
    {
        return true;
    }
    return false;
}
```

PackedStringArray **_get_breakpoints**() virtual const

This is for editors that edit script-based objects. You can return a list of breakpoints in the format (`script:line`), for example: `res://path_to_script.gd:25`.

Texture2D **_get_plugin_icon**() virtual const

Override this method in your plugin to return a Texture2D in order to give it an icon.

For main screen plugins, this appears at the top of the screen, to the right of the "2D", "3D", "Script", and "AssetLib" buttons.

Ideally, the plugin icon should be white with a transparent background and 16×16 pixels in size.

GDScript

```
func _get_plugin_icon():
    # You can use a custom icon:
    return preload("res://addons/my_plugin/my_plugin_icon.svg")
    # Or use a built-in icon:
    return EditorInterface.get_editor_theme().get_icon("Node", "EditorIcons")
```

C#

```
public override Texture2D _GetPluginIcon()
{
    // You can use a custom icon:
    return ResourceLoader.Load<Texture2D>
("res://addons/my_plugin/my_plugin_icon.svg");
    // Or use a built-in icon:
    return EditorInterface.Singleton.GetEditorTheme().GetIcon("Node",
"EditorIcons");
}
```

String **_get_plugin_name**() virtual const 🔗

Override this method in your plugin to provide the name of the plugin when displayed in the Godot editor.

For main screen plugins, this appears at the top of the screen, to the right of the "2D", "3D", "Script", and "AssetLib" buttons.

Dictionary **_get_state**() virtual const 🔗

Override this method to provide a state data you want to be saved, like view position, grid settings, folding, etc. This is used when saving the scene (so state is kept when opening it again) and for switching tabs (so state can be restored when the tab returns). This data is automatically saved for each scene in an `editstate` file in the editor metadata folder. If you want to store global (scene-independent) editor data for your plugin, you can use _get_window_layout instead.

Use _set_state to restore your saved state.

Note: This method should not be used to save important settings that should persist with the project.

Note: You must implement _get_plugin_name for the state to be stored and restored correctly.

```
func _get_state():
    var state = {"zoom": zoom, "preferred_color": my_color}
    return state
```

String **_get_unsaved_status**(for_scene: String) virtual const

Override this method to provide a custom message that lists unsaved changes. The editor will call this method when exiting or when closing a scene, and display the returned string in a confirmation dialog. Return empty string if the plugin has no unsaved changes.

When closing a scene, `for_scene` is the path to the scene being closed. You can use it to handle built-in resources in that scene.

If the user confirms saving, _save_external_data will be called, before closing the editor.

```
func _get_unsaved_status(for_scene):
    if not unsaved:
        return ""

    if for_scene.is_empty():
        return "Save changes in MyCustomPlugin before closing?"
    else:
        return "Scene %s has changes from MyCustomPlugin. Save before closing?" %
for_scene.get_file()

func _save_external_data():
    unsaved = false
```

If the plugin has no scene-specific changes, you can ignore the calls when closing scenes:

```
func _get_unsaved_status(for_scene):
    if not for_scene.is_empty():
        return ""
```

---

void **_get_window_layout**(configuration: ConfigFile) virtual

Override this method to provide the GUI layout of the plugin or any other data you want to be stored. This is used to save the project's editor layout when queue_save_layout is called or the editor layout was changed (for example changing the position of a dock). The data is stored in the `editor_layout.cfg` file in the editor metadata directory.

Use _set_window_layout to restore your saved layout.

```
func _get_window_layout(configuration):
    configuration.set_value("MyPlugin", "window_position", $Window.position)
    configuration.set_value("MyPlugin", "icon_color", $Icon.modulate)
```

---

bool **_handles**(object: Object) virtual const

Implement this function if your plugin edits a specific type of object (Resource or Node). If you return `true`, then you will get the functions _edit and _make_visible called when the editor requests them. If you have declared the methods _forward_canvas_gui_input and _forward_3d_gui_input these will be called too.

Note: Each plugin should handle only one type of objects at a time. If a plugin handles more types of objects and they are edited at the same time, it will result in errors.

---

bool **_has_main_screen**() virtual const

Returns `true` if this is a main screen editor plugin (it goes in the workspace selector together with **2D**, **3D**, **Script** and **AssetLib**).

When the plugin's workspace is selected, other main screen plugins will be hidden, but your plugin will not appear automatically. It needs to be added as a child of EditorInterface.get_editor_main_screen and made visible inside _make_visible.

Use _get_plugin_name and _get_plugin_icon to customize the plugin button's appearance.

```
var plugin_control

func _enter_tree():
    plugin_control = preload("my_plugin_control.tscn").instantiate()
    EditorInterface.get_editor_main_screen().add_child(plugin_control)
    plugin_control.hide()

func _has_main_screen():
    return true

func _make_visible(visible):
    plugin_control.visible = visible

func _get_plugin_name():
    return "My Super Cool Plugin 3000"

func _get_plugin_icon():
    return EditorInterface.get_editor_theme().get_icon("Node", "EditorIcons")
```

---

void **_make_visible**(visible: bool) virtual

This function will be called when the editor is requested to become visible. It is used for plugins that edit a specific object type.

Remember that you have to manage the visibility of all your editor controls manually.

---

void **_save_external_data**() virtual

This method is called after the editor saves the project or when it's closed. It asks the plugin to save edited external scenes/resources.

void **_set_state**(state: Dictionary) virtual

Restore the state saved by _get_state. This method is called when the current scene tab is changed in the editor.

Note: Your plugin must implement _get_plugin_name, otherwise it will not be recognized and this method will not be called.

```
func _set_state(data):
    zoom = data.get("zoom", 1.0)
    preferred_color = data.get("my_color", Color.WHITE)
```

void **_set_window_layout**(configuration: ConfigFile) virtual

Restore the plugin GUI layout and data saved by _get_window_layout. This method is called for every plugin on editor startup. Use the provided `configuration` file to read your saved data.

```
func _set_window_layout(configuration):
    $Window.position = configuration.get_value("MyPlugin", "window_position", Vector2())
    $Icon.modulate = configuration.get_value("MyPlugin", "icon_color", Color.WHITE)
```

void **add_autoload_singleton**(name: String, path: String)

Adds a script at `path` to the Autoload list as `name`.

Button **add_control_to_bottom_panel**(control: Control, title: String, shortcut: Shortcut = null)

Adds a control to the bottom panel (together with Output, Debug, Animation, etc). Returns a reference to the button added. It's up to you to hide/show the button when needed. When your plugin is deactivated, make sure to remove your custom control with remove_control_from_bottom_panel and free it with Node.queue_free.

Optionally, you can specify a shortcut parameter. When pressed, this shortcut will toggle the bottom panel's visibility. See the default editor bottom panel shortcuts in the Editor Settings for inspiration. Per convention, they all use `Alt` modifier.

void **add_control_to_container**(container: CustomControlContainer, control: Control)

Adds a custom control to a container (see CustomControlContainer). There are many locations where custom controls can be added in the editor UI.

Please remember that you have to manage the visibility of your custom controls yourself (and likely hide it after adding it).

When your plugin is deactivated, make sure to remove your custom control with remove_control_from_container and free it with Node.queue_free.

---

void **add_control_to_dock**(slot: DockSlot, control: Control, shortcut: Shortcut = null)

Adds the control to a specific dock slot (see DockSlot for options).

If the dock is repositioned and as long as the plugin is active, the editor will save the dock position on further sessions.

When your plugin is deactivated, make sure to remove your custom control with remove_control_from_docks and free it with Node.queue_free.

Optionally, you can specify a shortcut parameter. When pressed, this shortcut will toggle the dock's visibility once it's moved to the bottom panel (this shortcut does not affect the dock otherwise). See the default editor bottom panel shortcuts in the Editor Settings for inspiration. Per convention, they all use `Alt` modifier.

---

void **add_custom_type**(type: String, base: String, script: Script, icon: Texture2D)

Adds a custom type, which will appear in the list of nodes or resources.

When a given node or resource is selected, the base type will be instantiated (e.g. "Node3D", "Control", "Resource"), then the script will be loaded and set to this object.

Note: The base type is the base engine class which this type's class hierarchy inherits, not any custom type parent classes.

You can use the virtual method _handles to check if your custom object is being edited by checking the script or using the `is` keyword.

During run-time, this will be a simple object with a script so this function does not need to be called then.

Note: Custom types added this way are not true classes. They are just a helper to create a node with specific script.

---

void **add_debugger_plugin**(script: EditorDebuggerPlugin)

Adds a Script as debugger plugin to the Debugger. The script must extend EditorDebuggerPlugin.

void **add_export_plugin**(plugin: EditorExportPlugin)

Registers a new EditorExportPlugin. Export plugins are used to perform tasks when the project is being exported.

See add_inspector_plugin for an example of how to register a plugin.

void **add_import_plugin**(importer: EditorImportPlugin, first_priority: bool = false)

Registers a new EditorImportPlugin. Import plugins are used to import custom and unsupported assets as a custom Resource type.

If `first_priority` is `true`, the new import plugin is inserted first in the list and takes precedence over pre-existing plugins.

Note: If you want to import custom 3D asset formats use add_scene_format_importer_plugin instead.

See add_inspector_plugin for an example of how to register a plugin.

void **add_inspector_plugin**(plugin: EditorInspectorPlugin)

Registers a new EditorInspectorPlugin. Inspector plugins are used to extend EditorInspector and provide custom configuration tools for your object's properties.

Note: Always use remove_inspector_plugin to remove the registered EditorInspectorPlugin when your **EditorPlugin** is disabled to prevent leaks and an unexpected behavior.

GDScript

```
const MyInspectorPlugin = preload("res://addons/your_addon/path/to/your/script.gd")
var inspector_plugin = MyInspectorPlugin.new()

func _enter_tree():
    add_inspector_plugin(inspector_plugin)

func _exit_tree():
    remove_inspector_plugin(inspector_plugin)
```

void **add_node_3d_gizmo_plugin**(plugin: EditorNode3DGizmoPlugin)

Registers a new EditorNode3DGizmoPlugin. Gizmo plugins are used to add custom gizmos to the 3D preview viewport for a Node3D.

See add_inspector_plugin for an example of how to register a plugin.

---

void **add_resource_conversion_plugin**(plugin: EditorResourceConversionPlugin)

Registers a new EditorResourceConversionPlugin. Resource conversion plugins are used to add custom resource converters to the editor inspector.

See EditorResourceConversionPlugin for an example of how to create a resource conversion plugin.

---

void **add_scene_format_importer_plugin**(scene_format_importer: EditorSceneFormatImporter, first_priority: bool = false)

Registers a new EditorSceneFormatImporter. Scene importers are used to import custom 3D asset formats as scenes.

If `first_priority` is `true`, the new import plugin is inserted first in the list and takes precedence over pre-existing plugins.

---

void **add_scene_post_import_plugin**(scene_import_plugin: EditorScenePostImportPlugin, first_priority: bool = false)

Add a EditorScenePostImportPlugin. These plugins allow customizing the import process of 3D assets by adding new options to the import dialogs.

If `first_priority` is `true`, the new import plugin is inserted first in the list and takes precedence over pre-existing plugins.

---

void **add_tool_menu_item**(name: String, callable: Callable)

Adds a custom menu item to **Project > Tools** named `name`. When clicked, the provided `callable` will be called.

---

void **add_tool_submenu_item**(name: String, submenu: PopupMenu)

Adds a custom PopupMenu submenu under **Project > Tools >** `name`. Use remove_tool_menu_item on plugin clean up to remove the menu.

---

void **add_translation_parser_plugin**(parser: EditorTranslationParserPlugin)

Registers a custom translation parser plugin for extracting translatable strings from custom files.

---

void **add_undo_redo_inspector_hook_callback**(callable: Callable) 🔗

Hooks a callback into the undo/redo action creation when a property is modified in the inspector. This allows, for example, to save other properties that may be lost when a given property is modified.

The callback should have 4 arguments: Object `undo_redo`, Object `modified_object`, String `property` and Variant `new_value`. They are, respectively, the UndoRedo object used by the inspector, the currently modified object, the name of the modified property and the new value the property is about to take.

EditorInterface **get_editor_interface**() 🔗

Deprecated: EditorInterface is a global singleton and can be accessed directly by its name.

Returns the EditorInterface singleton instance.

PopupMenu **get_export_as_menu**() 🔗

Returns the PopupMenu under **Scene > Export As...**.

String **get_plugin_version**() const 🔗

Provide the version of the plugin declared in the `plugin.cfg` config file.

ScriptCreateDialog **get_script_create_dialog**() 🔗

Gets the Editor's dialog used for making scripts.

Note: Users can configure it before use.

Warning: Removing and freeing this node will render a part of the editor useless and may cause a crash.

EditorUndoRedoManager **get_undo_redo**() 🔗

Gets the undo/redo object. Most actions in the editor can be undoable, so use this object to make sure this happens when it's worth it.

---

void **hide_bottom_panel**()

Minimizes the bottom panel.

void **make_bottom_panel_item_visible**(item: Control)

Makes a specific item in the bottom panel visible.

void **queue_save_layout**()

Queue save the project's editor layout.

void **remove_autoload_singleton**(name: String)

Removes an Autoload `name` from the list.

void **remove_control_from_bottom_panel**(control: Control)

Removes the control from the bottom panel. You have to manually Node.queue_free the control.

void **remove_control_from_container**(container: CustomControlContainer, control: Control)

Removes the control from the specified container. You have to manually Node.queue_free the control.

void **remove_control_from_docks**(control: Control)

Removes the control from the dock. You have to manually Node.queue_free the control.

void **remove_custom_type**(type: String)

Removes a custom type added by add_custom_type.

void **remove_debugger_plugin**(script: EditorDebuggerPlugin)

Removes the debugger plugin with given script from the Debugger.

void **remove_export_plugin**(plugin: EditorExportPlugin) 🔗

Removes an export plugin registered by add_export_plugin.

---

void **remove_import_plugin**(importer: EditorImportPlugin) 🔗

Removes an import plugin registered by add_import_plugin.

---

void **remove_inspector_plugin**(plugin: EditorInspectorPlugin) 🔗

Removes an inspector plugin registered by add_import_plugin

---

void **remove_node_3d_gizmo_plugin**(plugin: EditorNode3DGizmoPlugin) 🔗

Removes a gizmo plugin registered by add_node_3d_gizmo_plugin.

---

void **remove_resource_conversion_plugin**(plugin: EditorResourceConversionPlugin) 🔗

Removes a resource conversion plugin registered by add_resource_conversion_plugin.

---

void **remove_scene_format_importer_plugin**(scene_format_importer: EditorSceneFormatImporter) 🔗

Removes a scene format importer registered by add_scene_format_importer_plugin.

---

void **remove_scene_post_import_plugin**(scene_import_plugin: EditorScenePostImportPlugin) 🔗

Remove the EditorScenePostImportPlugin, added with add_scene_post_import_plugin.

---

void **remove_tool_menu_item**(name: String) 🔗

Removes a menu `name` from **Project > Tools**.

---

void **remove_translation_parser_plugin**(parser: EditorTranslationParserPlugin) 🔗

Removes a custom translation parser plugin registered by add_translation_parser_plugin.

---

void **remove_undo_redo_inspector_hook_callback**(callable: Callable) 🔗

Removes a callback previously added by add_undo_redo_inspector_hook_callback.

---

void **set_dock_tab_icon**(control: Control, icon: Texture2D) 🔗

Sets the tab icon for the given control in a dock slot. Setting to `null` removes the icon.

---

void **set_force_draw_over_forwarding_enabled**() 🔗

Enables calling of _forward_canvas_force_draw_over_viewport for the 2D editor and _forward_3d_force_draw_over_viewport for the 3D editor when their viewports are updated. You need to call this method only once and it will work permanently for this plugin.

---

void **set_input_event_forwarding_always_enabled**() 🔗

Use this method if you always want to receive inputs from 3D view screen inside _forward_3d_gui_input. It might be especially usable if your plugin will want to use raycast in the scene.

---

int **update_overlays**() const 🔗

Updates the overlays of the 2D and 3D editor viewport. Causes methods _forward_canvas_draw_over_viewport, _forward_canvas_force_draw_over_viewport, _forward_3d_draw_over_viewport and _forward_3d_force_draw_over_viewport to be called.

EditorProperty

Inherits: Container **<** Control **<** CanvasItem **<** Node **<** Object

Custom control for editing properties that can be added to the EditorInspector.

Description

A custom control for editing properties that can be added to the EditorInspector. It is added via EditorInspectorPlugin.

Properties

bool	checkable	`false`
bool	checked	`false`
bool	deletable	`false`
bool	draw_warning	`false`
bool	keying	`false`
String	label	`""`

bool	read_only	`false`

Methods

void	_set_read_only(read_only: bool) virtual
void	_update_property() virtual
void	add_focusable(control: Control)
void	emit_changed(property: StringName, value: Variant, field: StringName = &"", changing: bool = false)
Object	get_edited_object()
StringName	get_edited_property() const
void	set_bottom_editor(editor: Control)
void	update_property()

Signals

multiple_properties_changed(properties: PackedStringArray, value: Array)

Emit it if you want multiple properties modified at the same time. Do not use if added via EditorInspectorPlugin._parse_property.

---

object_id_selected(property: StringName, id: int)

Used by sub-inspectors. Emit it if what was selected was an Object ID.

---

property_can_revert_changed(property: StringName, can_revert: bool)

Emitted when the revertability (i.e., whether it has a non-default value and thus is displayed with a revert icon) of a property has changed.

---

property_changed(property: StringName, value: Variant, field: StringName, changing: bool)

Do not emit this manually, use the emit_changed method instead.

---

property_checked(property: StringName, checked: bool)

Emitted when a property was checked. Used internally.

---

property_deleted(property: StringName)

Emitted when a property was deleted. Used internally.

---

property_keyed(property: StringName) 🔗

Emit it if you want to add this value as an animation key (check for keying being enabled first).

property_keyed_with_value(property: StringName, value: Variant) 🔗

Emit it if you want to key a property with a single value.

property_pinned(property: StringName, pinned: bool) 🔗

Emit it if you want to mark (or unmark) the value of a property for being saved regardless of being equal to the default value.

The default value is the one the property will get when the node is just instantiated and can come from an ancestor scene in the inheritance/instantiation chain, a script or a builtin class.

resource_selected(path: String, resource: Resource) 🔗

If you want a sub-resource to be edited, emit this signal with the resource.

selected(path: String, focusable_idx: int) 🔗

Emitted when selected. Used internally.

Property Descriptions

bool **checkable** = `false` 🔗

- void **set_checkable**(value: bool)
- bool **is_checkable**()

Used by the inspector, set to `true` when the property is checkable.

bool **checked** = `false`

- void **set_checked**(value: bool)
- bool **is_checked**()

Used by the inspector, set to `true` when the property is checked.

bool **deletable** = `false`

- void **set_deletable**(value: bool)
- bool **is_deletable**()

Used by the inspector, set to `true` when the property can be deleted by the user.

bool **draw_warning** = `false`

- void **set_draw_warning**(value: bool)
- bool **is_draw_warning**()

Used by the inspector, set to `true` when the property is drawn with the editor theme's warning color. This is used for editable children's properties.

bool **keying** = `false`

- void **set_keying**(value: bool)
- bool **is_keying**()

Used by the inspector, set to `true` when the property can add keys for animation.

String **label** = `""`

- void **set_label**(value: String)
- String **get_label**()

Set this property to change the label (if you want to show one).

---

bool **read_only** = `false`

- void **set_read_only**(value: bool)
- bool **is_read_only**()

Used by the inspector, set to `true` when the property is read-only.

Method Descriptions

void **_set_read_only**(read_only: bool) virtual

Called when the read-only status of the property is changed. It may be used to change custom controls into a read-only or modifiable state.

---

void **_update_property**() virtual

When this virtual function is called, you must update your editor.

---

void **add_focusable**(control: Control)

If any of the controls added can gain keyboard focus, add it here. This ensures that focus will be restored if the inspector is refreshed.

void **emit_changed**(property: StringName, value: Variant, field: StringName = &"", changing: bool = false)

If one or several properties have changed, this must be called. `field` is used in case your editor can modify fields separately (as an example, Vector3.x). The `changing` argument avoids the editor requesting this property to be refreshed (leave as `false` if unsure).

Object **get_edited_object**()

Gets the edited object.

StringName **get_edited_property**() const

Gets the edited property. If your editor is for a single property (added via EditorInspectorPlugin._parse_property), then this will return the property.

void **set_bottom_editor**(editor: Control)

Puts the `editor` control below the property label. The control must be previously added using Node.add_child.

void **update_property**()

Forces refresh of the property display.

EditorResourcePicker

Inherits: HBoxContainer **<** BoxContainer **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

Inherited By: EditorScriptPicker

Godot editor's control for selecting Resource type properties.

Description

This Control node is used in the editor's Inspector dock to allow editing of Resource type properties. It provides options for creating, loading, saving and converting resources. Can be used with EditorInspectorPlugin to recreate the same behavior.

Note: This Control does not include any editor for the resource, as editing is controlled by the Inspector dock itself or sub-Inspectors.

Properties

String	base_type	`""`
bool	editable	`true`
Resource	edited_resource	
bool	toggle_mode	`false`

Methods

bool	_handle_menu_selected(id: int) virtual
void	_set_create_options(menu_node: Object) virtual
PackedStringArray	get_allowed_types() const
void	set_toggle_pressed(pressed: bool)

Signals

resource_changed(resource: Resource) 🔗

Emitted when the value of the edited resource was changed.

---

resource_selected(resource: Resource, inspect: bool) 🔗

Emitted when the resource value was set and user clicked to edit it. When `inspect` is `true`, the signal was caused by the context menu "Edit" or "Inspect" option.

Property Descriptions

String **base_type** = `""` 🔗

- void **set_base_type**(value: String)
- String **get_base_type**()

The base type of allowed resource types. Can be a comma-separated list of several options.

---

bool **editable** = `true`

- void **set_editable**(value: bool)
- bool **is_editable**()

If `true`, the value can be selected and edited.

Resource **edited_resource**

- void **set_edited_resource**(value: Resource)
- Resource **get_edited_resource**()

The edited resource value.

---

bool **toggle_mode** = `false`

- void **set_toggle_mode**(value: bool)
- bool **is_toggle_mode**()

If `true`, the main button with the resource preview works in the toggle mode. Use set_toggle_pressed to manually set the state.

Method Descriptions

bool **_handle_menu_selected**(id: int) virtual

This virtual method can be implemented to handle context menu items not handled by default. See _set_create_options.

---

void **_set_create_options**(menu_node: Object) virtual 🔗

This virtual method is called when updating the context menu of **EditorResourcePicker**. Implement this method to override the "New ..." items with your own options. `menu_node` is a reference to the PopupMenu node.

Note: Implement _handle_menu_selected to handle these custom items.

PackedStringArray **get_allowed_types**() const 🔗

Returns a list of all allowed types and subtypes corresponding to the base_type. If the base_type is empty, an empty list is returned.

---

void **set_toggle_pressed**(pressed: bool) 🔗

Sets the toggle mode state for the main button. Works only if toggle_mode is set to `true`.

EditorResourcePreview

Inherits: Node **<** Object

A node used to generate previews of resources or files.

Description

This node is used to generate previews for resources or files.

Note: This class shouldn't be instantiated directly. Instead, access the singleton using EditorInterface.get_resource_previewer.

Methods

void	add_preview_generator(generator: EditorResourcePreviewGenerator)
void	check_for_invalidation(path: String)
void	queue_edited_resource_preview(resource: Resource, receiver: Object, receiver_func: StringName, userdata: Variant)
void	queue_resource_preview(path: String, receiver: Object, receiver_func: StringName, userdata: Variant)

void remove_preview_generator(generator: EditorResourcePreviewGenerator)

Signals

preview_invalidated(path: String) 🔗

Emitted if a preview was invalidated (changed). `path` corresponds to the path of the preview.

Method Descriptions

void **add_preview_generator**(generator: EditorResourcePreviewGenerator) 🔗

Create an own, custom preview generator.

void **check_for_invalidation**(path: String) 🔗

Check if the resource changed, if so, it will be invalidated and the corresponding signal emitted.

void **queue_edited_resource_preview**(resource: Resource, receiver: Object, receiver_func: StringName, userdata: Variant) 🔗

Queue the `resource` being edited for preview. Once the preview is ready, the `receiver`'s `receiver_func` will be called. The `receiver_func` must take the following four arguments: String path, Texture2D preview, Texture2D thumbnail_preview, Variant userdata. `userdata` can be anything, and will be returned when `receiver_func` is called.

Note: If it was not possible to create the preview the `receiver_func` will still be called, but the preview will be null.

void **queue_resource_preview**(path: String, receiver: Object, receiver_func: StringName, userdata: Variant) 🔗

Queue a resource file located at `path` for preview. Once the preview is ready, the `receiver`'s `receiver_func` will be called. The `receiver_func` must take the following four arguments: String path, Texture2D preview, Texture2D thumbnail_preview, Variant userdata. `userdata` can be anything, and will be returned when `receiver_func` is called.

Note: If it was not possible to create the preview the `receiver_func` will still be called, but the preview will be null.

void **remove_preview_generator**(generator: EditorResourcePreviewGenerator) 🔗

Removes a custom preview generator.

EditorScriptPicker

Inherits: EditorResourcePicker **<** HBoxContainer **<** BoxContainer **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

Godot editor's control for selecting the `script` property of a Node.

Description

Similar to EditorResourcePicker this Control node is used in the editor's Inspector dock, but only to edit the `script` property of a Node. Default options for creating new resources of all possible subtypes are replaced with dedicated buttons that open the "Attach Node Script" dialog. Can be used with EditorInspectorPlugin to recreate the same behavior.

Note: You must set the script_owner for the custom context menu items to work.

Properties

Node	script_owner

Property Descriptions

Node **script_owner** 🔗

- void **set_script_owner**(value: Node)

- Node **get_script_owner**()

The owner Node of the script property that holds the edited resource.

EditorSpinSlider

Inherits: Range **<** Control **<** CanvasItem **<** Node **<** Object

Godot editor's control for editing numeric values.

Description

This Control node is used in the editor's Inspector dock to allow editing of numeric values. Can be used with EditorInspectorPlugin to recreate the same behavior.

If the Range.step value is `1`, the **EditorSpinSlider** will display up/down arrows, similar to SpinBox. If the Range.step value is not `1`, a slider will be displayed instead.

Properties

bool	flat	`false`
FocusMode	focus_mode	`2` (overrides Control)
bool	hide_slider	`false`
String	label	`""`
bool	read_only	`false`

BitField[SizeFlags]	size_flags_vertical	`1` (overrides Control)
float	step	`1.0` (overrides Range)
String	suffix	`""`

Signals

grabbed() 🔗

Emitted when the spinner/slider is grabbed.

ungrabbed() 🔗

Emitted when the spinner/slider is ungrabbed.

value_focus_entered() 🔗

Emitted when the value form gains focus.

value_focus_exited() 🔗

Emitted when the value form loses focus.

Property Descriptions

bool **flat** = `false`

- void **set_flat**(value: bool)
- bool **is_flat**()

If `true`, the slider will not draw background.

---

bool **hide_slider** = `false`

- void **set_hide_slider**(value: bool)
- bool **is_hiding_slider**()

If `true`, the slider and up/down arrows are hidden.

String **label** = `""`

- void **set_label**(value: String)
- String **get_label**()

The text that displays to the left of the value.

---

bool **read_only** = `false`

- void **set_read_only**(value: bool)
- bool **is_read_only**()

If `true`, the slider can't be interacted with.

String **suffix** = `""`

- void **set_suffix**(value: String)
- String **get_suffix**()

The suffix to display after the value (in a faded color). This should generally be a plural word. You may have to use an abbreviation if the suffix is too long to be displayed.

FileDialog

Inherits: ConfirmationDialog **<** AcceptDialog **<** Window **<** Viewport **<** Node **<** Object

A dialog for selecting files or directories in the filesystem.

Description

FileDialog is a preset dialog used to choose files and directories in the filesystem. It supports filter masks. **FileDialog** automatically sets its window title according to the file_mode. If you want to use a custom title, disable this by setting mode_overrides_title to `false`.

Properties

Access	access	`0`
String	current_dir	
String	current_file	
String	current_path	
bool	dialog_hide_on_ok	`false` (overrides AcceptDialog)

FileMode	file_mode	`4`
PackedStringArray	filters	`PackedStringArray()`
bool	mode_overrides_title	`true`
int	option_count	`0`
String	root_subfolder	`""`
bool	show_hidden_files	`false`
String	title	`"Save a File"` (overrides Window)
bool	use_native_dialog	`false`

Methods

void	add_filter(filter: String, description: String = "")
void	add_option(name: String, values: PackedStringArray, default_value_index: int)

void	clear_filters()
void	deselect_all()
LineEdit	get_line_edit()
int	get_option_default(option: int) const
String	get_option_name(option: int) const
PackedStringArray	get_option_values(option: int) const
Dictionary	get_selected_options() const
VBoxContainer	get_vbox()
void	invalidate()
void	set_option_default(option: int, default_value_index: int)
void	set_option_name(option: int, name: String)
void	set_option_values(option: int, values: PackedStringArray)

Theme Properties

Color	file_disabled_color	`Color(1, 1, 1, 0.25)`
Color	file_icon_color	`Color(1, 1, 1, 1)`
Color	folder_icon_color	`Color(1, 1, 1, 1)`
Texture2D	back_folder	
Texture2D	create_folder	
Texture2D	file	
Texture2D	folder	
Texture2D	forward_folder	
Texture2D	parent_folder	
Texture2D	reload	
Texture2D	toggle_hidden	

Signals

dir_selected(dir: String)

Emitted when the user selects a directory.

---

file_selected(path: String)

Emitted when the user selects a file by double-clicking it or pressing the **OK** button.

---

files_selected(paths: PackedStringArray)

Emitted when the user selects multiple files.

Enumerations

enum **FileMode**:

FileMode **FILE_MODE_OPEN_FILE** = `0`

The dialog allows selecting one, and only one file.

FileMode **FILE_MODE_OPEN_FILES** = `1`

The dialog allows selecting multiple files.

FileMode **FILE_MODE_OPEN_DIR** = `2`

The dialog only allows selecting a directory, disallowing the selection of any file.

FileMode **FILE_MODE_OPEN_ANY** = `3`

The dialog allows selecting one file or directory.

FileMode **FILE_MODE_SAVE_FILE** = `4`

The dialog will warn when a file exists.

enum **Access**: 🔗

Access **ACCESS_RESOURCES** = `0`

The dialog only allows accessing files under the Resource path (`res://`).

Access **ACCESS_USERDATA** = `1`

The dialog only allows accessing files under user data path (`user://`).

Access **ACCESS_FILESYSTEM** = `2`

The dialog allows accessing files on the whole file system.

Property Descriptions

Access **access** = `0` 🔗

- void **set_access**(value: Access)
- Access **get_access**()

The file system access scope. See Access constants.

Warning: In Web builds, FileDialog cannot access the host file system. In sandboxed Linux and macOS environments, use_native_dialog is automatically used to allow limited access to host file system.

String **current_dir**

- void **set_current_dir**(value: String)
- String **get_current_dir**()

The current working directory of the file dialog.

Note: For native file dialogs, this property is only treated as a hint and may not be respected by specific OS implementations.

String **current_file**

- void **set_current_file**(value: String)
- String **get_current_file**()

The currently selected file of the file dialog.

String **current_path**

- void **set_current_path**(value: String)
- String **get_current_path**()

The currently selected file path of the file dialog.

FileMode **file_mode** = `4`

- void **set_file_mode**(value: FileMode)
- FileMode **get_file_mode**()

The dialog's open or save mode, which affects the selection behavior. See FileMode.

PackedStringArray **filters** = `PackedStringArray()` 🔗

- void **set_filters**(value: PackedStringArray)
- PackedStringArray **get_filters**()

The available file type filters. Each filter string in the array should be formatted like this: `*.txt,*.doc;Text Files`. The description text of the filter is optional and can be omitted.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedStringArray for more details.

---

bool **mode_overrides_title** = `true` 🔗

- void **set_mode_overrides_title**(value: bool)
- bool **is_mode_overriding_title**()

If `true`, changing the file_mode property will set the window title accordingly (e.g. setting file_mode to FILE_MODE_OPEN_FILE will change the window title to "Open a File").

---

int **option_count** = `0` 🔗

- void **set_option_count**(value: int)
- int **get_option_count**()

The number of additional OptionButtons and CheckBoxes in the dialog.

String **root_subfolder** = `""`

- void **set_root_subfolder**(value: String)
- String **get_root_subfolder**()

If non-empty, the given sub-folder will be "root" of this **FileDialog**, i.e. user won't be able to go to its parent directory.

Note: This property is ignored by native file dialogs.

---

bool **show_hidden_files** = `false`

- void **set_show_hidden_files**(value: bool)
- bool **is_showing_hidden_files**()

If `true`, the dialog will show hidden files.

Note: This property is ignored by native file dialogs on Linux.

---

bool **use_native_dialog** = `false`

- void **set_use_native_dialog**(value: bool)
- bool **get_use_native_dialog**()

If `true`, access is set to ACCESS_FILESYSTEM, and it is supported by the current DisplayServer, OS native dialog will be used instead of custom one.

Note: On Linux and macOS, sandboxed apps always use native dialogs to access the host file system.

Note: On macOS, sandboxed apps will save security-scoped bookmarks to retain access to the opened folders across multiple sessions. Use OS.get_granted_permissions to get a list of saved bookmarks.

Note: Native dialogs are isolated from the base process, file dialog properties can't be modified once the dialog is shown.

Method Descriptions

void **add_filter**(filter: String, description: String = "")

Adds a comma-delimited file name `filter` option to the **FileDialog** with an optional `description`, which restricts what files can be picked.

A `filter` should be of the form `"filename.extension"`, where filename and extension can be `*` to match any string. Filters starting with `.` (i.e. empty filenames) are not allowed.

For example, a `filter` of `"*.png, *.jpg"` and a `description` of `"Images"` results in filter text "Images (*.png, *.jpg)".

void **add_option**(name: String, values: PackedStringArray, default_value_index: int)

Adds an additional OptionButton to the file dialog. If `values` is empty, a CheckBox is added instead.

`default_value_index` should be an index of the value in the `values`. If `values` is empty it should be either `1` (checked), or `0` (unchecked).

void **clear_filters**()

Clear all the added filters in the dialog.

void **deselect_all**()

Clear all currently selected items in the dialog.

LineEdit **get_line_edit**()

Returns the LineEdit for the selected file.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their CanvasItem.visible property.

---

int **get_option_default**(option: int) const

Returns the default value index of the OptionButton or CheckBox with index `option`.

String **get_option_name**(option: int) const

Returns the name of the OptionButton or CheckBox with index `option`.

PackedStringArray **get_option_values**(option: int) const

Returns an array of values of the OptionButton with index `option`.

Dictionary **get_selected_options**() const

Returns a Dictionary with the selected values of the additional OptionButtons and/or CheckBoxes. Dictionary keys are names and values are selected value indices.

VBoxContainer **get_vbox**()

Returns the vertical box container of the dialog, custom controls can be added to it.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their CanvasItem.visible property.

Note: Changes to this node are ignored by native file dialogs, use add_option to add custom elements to the dialog instead.

---

void **invalidate**()

Invalidate and update the current dialog content list.

Note: This method does nothing on native file dialogs.

---

void **set_option_default**(option: int, default_value_index: int)

Sets the default value index of the OptionButton or CheckBox with index `option`.

---

void **set_option_name**(option: int, name: String)

Sets the name of the OptionButton or CheckBox with index `option`.

---

void **set_option_values**(option: int, values: PackedStringArray)

Sets the option values of the OptionButton with index `option`.

Theme Property Descriptions

Color **file_disabled_color** = `Color(1, 1, 1, 0.25)`

The color tint for disabled files (when the **FileDialog** is used in open folder mode).

Color **file_icon_color** = `Color(1, 1, 1, 1)`

The color modulation applied to the file icon.

Color **folder_icon_color** = `Color(1, 1, 1, 1)`

The color modulation applied to the folder icon.

Texture2D **back_folder**

Custom icon for the back arrow.

Texture2D **create_folder**

Custom icon for the create folder button.

Texture2D **file**

Custom icon for files.

Texture2D **folder**

Custom icon for folders.

Texture2D **forward_folder**

Custom icon for the forward arrow.

Texture2D **parent_folder** 🔗

Custom icon for the parent folder arrow.

Texture2D **reload** 🔗

Custom icon for the reload button.

Texture2D **toggle_hidden** 🔗

Custom icon for the toggle hidden button.

FileSystemDock

Inherits: VBoxContainer **<** BoxContainer **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

Godot editor's dock for managing files in the project.

Description

This class is available only in EditorPlugins and can't be instantiated. You can access it using EditorInterface.get_file_system_dock.

While **FileSystemDock** doesn't expose any methods for file manipulation, it can listen for various file-related signals.

Methods

void	add_resource_tooltip_plugin(plugin: EditorResourceTooltipPlugin)
void	navigate_to_path(path: String)
void	remove_resource_tooltip_plugin(plugin: EditorResourceTooltipPlugin)

Signals

display_mode_changed()

Emitted when the user switches file display mode or split mode.

---

file_removed(file: String)

Emitted when the given `file` was removed.

---

files_moved(old_file: String, new_file: String)

Emitted when a file is moved from `old_file` path to `new_file` path.

---

folder_color_changed()

Emitted when folders change color.

---

folder_moved(old_folder: String, new_folder: String)

Emitted when a folder is moved from `old_folder` path to `new_folder` path.

---

folder_removed(folder: String)

Emitted when the given `folder` was removed.

---

inherit(file: String)

Emitted when a new scene is created that inherits the scene at `file` path.

---

instantiate(files: PackedStringArray)

Emitted when the given scenes are being instantiated in the editor.

---

resource_removed(resource: Resource)

Emitted when an external `resource` had its file removed.

Method Descriptions

void **add_resource_tooltip_plugin**(plugin: EditorResourceTooltipPlugin)

Registers a new EditorResourceTooltipPlugin.

---

void **navigate_to_path**(path: String)

Sets the given `path` as currently selected, ensuring that the selected file/directory is visible.

---

void **remove_resource_tooltip_plugin**(plugin: EditorResourceTooltipPlugin)

Removes an EditorResourceTooltipPlugin. Fails if the plugin wasn't previously added.

FlowContainer

Inherits: Container **<** Control **<** CanvasItem **<** Node **<** Object

Inherited By: HFlowContainer, VFlowContainer

A container that arranges its child controls horizontally or vertically and wraps them around at the borders.

Description

A container that arranges its child controls horizontally or vertically and wraps them around at the borders. This is similar to how text in a book wraps around when no more words can fit on a line.

Tutorials

- Using Containers

Properties

AlignmentMode	alignment	`0`
LastWrapAlignmentMode	last_wrap_alignment	`0`
bool	reverse_fill	`false`

bool	vertical	`false`

Methods

int	get_line_count() const

Theme Properties

int	h_separation	`4`
int	v_separation	`4`

Enumerations

enum **AlignmentMode**: 🔗

AlignmentMode **ALIGNMENT_BEGIN** = `0`

The child controls will be arranged at the beginning of the container, i.e. top if orientation is vertical, left if orientation is horizontal (right for RTL layout).

AlignmentMode **ALIGNMENT_CENTER** = `1`

The child controls will be centered in the container.

AlignmentMode **ALIGNMENT_END** = `2`

The child controls will be arranged at the end of the container, i.e. bottom if orientation is vertical, right if orientation is horizontal (left for RTL layout).

---

enum **LastWrapAlignmentMode**:

LastWrapAlignmentMode **LAST_WRAP_ALIGNMENT_INHERIT** = `0`

The last partially filled row or column will wrap aligned to the previous row or column in accordance with alignment.

LastWrapAlignmentMode **LAST_WRAP_ALIGNMENT_BEGIN** = `1`

The last partially filled row or column will wrap aligned to the beginning of the previous row or column.

LastWrapAlignmentMode **LAST_WRAP_ALIGNMENT_CENTER** = `2`

The last partially filled row or column will wrap aligned to the center of the previous row or column.

LastWrapAlignmentMode **LAST_WRAP_ALIGNMENT_END** = `3`

The last partially filled row or column will wrap aligned to the end of the previous row or column.

Property Descriptions

AlignmentMode **alignment** = `0`

- void **set_alignment**(value: AlignmentMode)
- AlignmentMode **get_alignment**()

The alignment of the container's children (must be one of ALIGNMENT_BEGIN, ALIGNMENT_CENTER, or ALIGNMENT_END).

LastWrapAlignmentMode **last_wrap_alignment** = `0` 🔗

- void **set_last_wrap_alignment**(value: LastWrapAlignmentMode)
- LastWrapAlignmentMode **get_last_wrap_alignment**()

The wrap behavior of the last, partially filled row or column (must be one of LAST_WRAP_ALIGNMENT_INHERIT, LAST_WRAP_ALIGNMENT_BEGIN, LAST_WRAP_ALIGNMENT_CENTER, or LAST_WRAP_ALIGNMENT_END).

---

bool **reverse_fill** = `false` 🔗

- void **set_reverse_fill**(value: bool)
- bool **is_reverse_fill**()

If `true`, reverses fill direction. Horizontal **FlowContainer**s will fill rows bottom to top, vertical **FlowContainer**s will fill columns right to left.

When using a vertical **FlowContainer** with a right to left Control.layout_direction, columns will fill left to right instead.

---

bool **vertical** = `false` 🔗

- void **set_vertical**(value: bool)
- bool **is_vertical**()

If `true`, the **FlowContainer** will arrange its children vertically, rather than horizontally.

Can't be changed when using HFlowContainer and VFlowContainer.

Method Descriptions

int **get_line_count**() const

Returns the current line count.

Theme Property Descriptions

int **h_separation** = `4`

The horizontal separation of child nodes.

---

int **v_separation** = `4`

The vertical separation of child nodes.

FogVolume

Inherits: VisualInstance3D **<** Node3D **<** Node **<** Object

A region that contributes to the default volumetric fog from the world environment.

Description

FogVolumes are used to add localized fog into the global volumetric fog effect. **FogVolume**s can also remove volumetric fog from specific areas if using a FogMaterial with a negative FogMaterial.density.

Performance of **FogVolume**s is directly related to their relative size on the screen and the complexity of their attached FogMaterial. It is best to keep **FogVolume**s relatively small and simple where possible.

Note: FogVolumes only have a visible effect if Environment.volumetric_fog_enabled is `true`. If you don't want fog to be globally visible (but only within **FogVolume** nodes), set Environment.volumetric_fog_density to `0.0`.

Tutorials

- Volumetric fog and fog volumes

Properties

Material	material	
FogVolumeShape	shape	`3`
Vector3	size	`Vector3(2, 2, 2)`

Property Descriptions

Material **material**

- void **set_material**(value: Material)
- Material **get_material**()

The Material used by the **FogVolume**. Can be either a built-in FogMaterial or a custom ShaderMaterial.

FogVolumeShape **shape** = `3`

- void **set_shape**(value: FogVolumeShape)
- FogVolumeShape **get_shape**()

The shape of the **FogVolume**. This can be set to either RenderingServer.FOG_VOLUME_SHAPE_ELLIPSOID, RenderingServer.FOG_VOLUME_SHAPE_CONE, RenderingServer.FOG_VOLUME_SHAPE_CYLINDER, RenderingServer.FOG_VOLUME_SHAPE_BOX or RenderingServer.FOG_VOLUME_SHAPE_WORLD.

Vector3 **size** = `Vector3(2, 2, 2)`

- void **set_size**(value: Vector3)
- Vector3 **get_size**()

The size of the **FogVolume** when shape is RenderingServer.FOG_VOLUME_SHAPE_ELLIPSOID, RenderingServer.FOG_VOLUME_SHAPE_CONE, RenderingServer.FOG_VOLUME_SHAPE_CYLINDER or RenderingServer.FOG_VOLUME_SHAPE_BOX.

Note: Thin fog volumes may appear to flicker when the camera moves or rotates. This can be alleviated by increasing ProjectSettings.rendering/environment/volumetric_fog/volume_depth (at a performance cost) or by decreasing Environment.volumetric_fog_length (at no performance cost, but at the cost of lower fog range). Alternatively, the **FogVolume** can be made thicker and use a lower density in the material.

Note: If shape is RenderingServer.FOG_VOLUME_SHAPE_CONE or RenderingServer.FOG_VOLUME_SHAPE_CYLINDER, the cone/cylinder will be adjusted to fit within the size. Non-uniform scaling of cone/cylinder shapes via the size property is not supported, but you can scale the **FogVolume** node instead.

Generic6DOFJoint3D

Inherits: Joint3D **<** Node3D **<** Node **<** Object

A physics joint that allows for complex movement and rotation between two 3D physics bodies.

Description

The **Generic6DOFJoint3D** (6 Degrees Of Freedom) joint allows for implementing custom types of joints by locking the rotation and translation of certain axes.

The first 3 DOF represent the linear motion of the physics bodies and the last 3 DOF represent the angular motion of the physics bodies. Each axis can be either locked, or limited.

Properties

float	angular_limit_x/damping	`1.0`
bool	angular_limit_x/enabled	`true`
float	angular_limit_x/erp	`0.5`
float	angular_limit_x/force_limit	`0.0`

float	angular_limit_x/lower_angle	`0.0`
float	angular_limit_x/restitution	`0.0`
float	angular_limit_x/softness	`0.5`
float	angular_limit_x/upper_angle	`0.0`
float	angular_limit_y/damping	`1.0`
bool	angular_limit_y/enabled	`true`
float	angular_limit_y/erp	`0.5`
float	angular_limit_y/force_limit	`0.0`
float	angular_limit_y/lower_angle	`0.0`
float	angular_limit_y/restitution	`0.0`
float	angular_limit_y/softness	`0.5`
float	angular_limit_y/upper_angle	`0.0`
float	angular_limit_z/damping	`1.0`

bool	angular_limit_z/enabled	`true`
float	angular_limit_z/erp	`0.5`
float	angular_limit_z/force_limit	`0.0`
float	angular_limit_z/lower_angle	`0.0`
float	angular_limit_z/restitution	`0.0`
float	angular_limit_z/softness	`0.5`
float	angular_limit_z/upper_angle	`0.0`
bool	angular_motor_x/enabled	`false`
float	angular_motor_x/force_limit	`300.0`
float	angular_motor_x/target_velocity	`0.0`
bool	angular_motor_y/enabled	`false`
float	angular_motor_y/force_limit	`300.0`

float	angular_motor_y/target_velocity	`0.0`
bool	angular_motor_z/enabled	`false`
float	angular_motor_z/force_limit	`300.0`
float	angular_motor_z/target_velocity	`0.0`
float	angular_spring_x/damping	`0.0`
bool	angular_spring_x/enabled	`false`
float	angular_spring_x/equilibrium_point	`0.0`
float	angular_spring_x/stiffness	`0.0`
float	angular_spring_y/damping	`0.0`
bool	angular_spring_y/enabled	`false`
float	angular_spring_y/equilibrium_point	`0.0`
float	angular_spring_y/stiffness	`0.0`
float	angular_spring_z/damping	`0.0`

bool	angular_spring_z/enabled	`false`
float	angular_spring_z/equilibrium_point	`0.0`
float	angular_spring_z/stiffness	`0.0`
float	linear_limit_x/damping	`1.0`
bool	linear_limit_x/enabled	`true`
float	linear_limit_x/lower_distance	`0.0`
float	linear_limit_x/restitution	`0.5`
float	linear_limit_x/softness	`0.7`
float	linear_limit_x/upper_distance	`0.0`
float	linear_limit_y/damping	`1.0`
bool	linear_limit_y/enabled	`true`
float	linear_limit_y/lower_distance	`0.0`

float	linear_limit_y/restitution	`0.5`
float	linear_limit_y/softness	`0.7`
float	linear_limit_y/upper_distance	`0.0`
float	linear_limit_z/damping	`1.0`
bool	linear_limit_z/enabled	`true`
float	linear_limit_z/lower_distance	`0.0`
float	linear_limit_z/restitution	`0.5`
float	linear_limit_z/softness	`0.7`
float	linear_limit_z/upper_distance	`0.0`
bool	linear_motor_x/enabled	`false`
float	linear_motor_x/force_limit	`0.0`
float	linear_motor_x/target_velocity	`0.0`
bool	linear_motor_y/enabled	`false`

float	linear_motor_y/force_limit	`0.0`
float	linear_motor_y/target_velocity	`0.0`
bool	linear_motor_z/enabled	`false`
float	linear_motor_z/force_limit	`0.0`
float	linear_motor_z/target_velocity	`0.0`
float	linear_spring_x/damping	`0.01`
bool	linear_spring_x/enabled	`false`
float	linear_spring_x/equilibrium_point	`0.0`
float	linear_spring_x/stiffness	`0.01`
float	linear_spring_y/damping	`0.01`
bool	linear_spring_y/enabled	`false`
float	linear_spring_y/equilibrium_point	`0.0`

float	linear_spring_y/stiffness	`0.01`
float	linear_spring_z/damping	`0.01`
bool	linear_spring_z/enabled	`false`
float	linear_spring_z/equilibrium_point	`0.0`
float	linear_spring_z/stiffness	`0.01`

Methods

bool	get_flag_x(flag: Flag) const
bool	get_flag_y(flag: Flag) const
bool	get_flag_z(flag: Flag) const
float	get_param_x(param: Param) const
float	get_param_y(param: Param) const
float	get_param_z(param: Param) const

void set_flag_x(flag: Flag, value: bool)

void set_flag_y(flag: Flag, value: bool)

void set_flag_z(flag: Flag, value: bool)

void set_param_x(param: Param, value: float)

void set_param_y(param: Param, value: float)

void set_param_z(param: Param, value: float)

Enumerations

enum **Param**: 🔗

Param **PARAM_LINEAR_LOWER_LIMIT** = `0`

The minimum difference between the pivot points' axes.

Param **PARAM_LINEAR_UPPER_LIMIT** = `1`

The maximum difference between the pivot points' axes.

Param **PARAM_LINEAR_LIMIT_SOFTNESS** = `2`

A factor applied to the movement across the axes. The lower, the slower the movement.

Param **PARAM_LINEAR_RESTITUTION** = `3`

The amount of restitution on the axes' movement. The lower, the more momentum gets lost.

Param **PARAM_LINEAR_DAMPING** = `4`

The amount of damping that happens at the linear motion across the axes.

Param **PARAM_LINEAR_MOTOR_TARGET_VELOCITY** = `5`

The velocity the linear motor will try to reach.

Param **PARAM_LINEAR_MOTOR_FORCE_LIMIT** = `6`

The maximum force the linear motor will apply while trying to reach the velocity target.

Param **PARAM_LINEAR_SPRING_STIFFNESS** = `7`

There is currently no description for this enum. Please help us by contributing one!

Param **PARAM_LINEAR_SPRING_DAMPING** = `8`

There is currently no description for this enum. Please help us by contributing one!

Param **PARAM_LINEAR_SPRING_EQUILIBRIUM_POINT** = `9`

There is currently no description for this enum. Please help us by contributing one!

Param **PARAM_ANGULAR_LOWER_LIMIT** = `10`

The minimum rotation in negative direction to break loose and rotate around the axes.

Param **PARAM_ANGULAR_UPPER_LIMIT** = `11`

The minimum rotation in positive direction to break loose and rotate around the axes.

Param **PARAM_ANGULAR_LIMIT_SOFTNESS** = `12`

The speed of all rotations across the axes.

Param **PARAM_ANGULAR_DAMPING** = `13`

The amount of rotational damping across the axes. The lower, the more damping occurs.

Param **PARAM_ANGULAR_RESTITUTION** = `14`

The amount of rotational restitution across the axes. The lower, the more restitution occurs.

Param **PARAM_ANGULAR_FORCE_LIMIT** = `15`

The maximum amount of force that can occur, when rotating around the axes.

Param **PARAM_ANGULAR_ERP** = `16`

When rotating across the axes, this error tolerance factor defines how much the correction gets slowed down. The lower, the slower.

Param **PARAM_ANGULAR_MOTOR_TARGET_VELOCITY** = `17`

Target speed for the motor at the axes.

Param **PARAM_ANGULAR_MOTOR_FORCE_LIMIT** = `18`

Maximum acceleration for the motor at the axes.

Param **PARAM_ANGULAR_SPRING_STIFFNESS** = `19`

There is currently no description for this enum. Please help us by contributing one!

Param **PARAM_ANGULAR_SPRING_DAMPING** = `20`

There is currently no description for this enum. Please help us by contributing one!

Param **PARAM_ANGULAR_SPRING_EQUILIBRIUM_POINT** = `21`

There is currently no description for this enum. Please help us by contributing one!

Param **PARAM_MAX** = `22`

Represents the size of the Param enum.

enum **Flag**: 🔗

Flag **FLAG_ENABLE_LINEAR_LIMIT** = `0`

If enabled, linear motion is possible within the given limits.

Flag **FLAG_ENABLE_ANGULAR_LIMIT** = `1`

If enabled, rotational motion is possible within the given limits.

Flag **FLAG_ENABLE_LINEAR_SPRING** = `3`

There is currently no description for this enum. Please help us by contributing one!

Flag **FLAG_ENABLE_ANGULAR_SPRING** = `2`

There is currently no description for this enum. Please help us by contributing one!

Flag **FLAG_ENABLE_MOTOR** = `4`

If enabled, there is a rotational motor across these axes.

Flag **FLAG_ENABLE_LINEAR_MOTOR** = `5`

If enabled, there is a linear motor across these axes.

Flag **FLAG_MAX** = `6`

Represents the size of the Flag enum.

Property Descriptions

float **angular_limit_x/damping** = `1.0`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

The amount of rotational damping across the X axis.

The lower, the longer an impulse from one side takes to travel to the other side.

bool **angular_limit_x/enabled** = `true`

- void **set_flag_x**(flag: Flag, value: bool)
- bool **get_flag_x**(flag: Flag) const

If `true`, rotation across the X axis is limited.

float **angular_limit_x/erp** = `0.5`

- void **set_param_x**(param: Param, value: float)

- float **get_param_x**(param: Param) const

When rotating across the X axis, this error tolerance factor defines how much the correction gets slowed down. The lower, the slower.

---

float **angular_limit_x/force_limit** = `0.0`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

The maximum amount of force that can occur, when rotating around the X axis.

---

float **angular_limit_x/lower_angle** = `0.0`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

The minimum rotation in negative direction to break loose and rotate around the X axis.

---

float **angular_limit_x/restitution** = `0.0`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

The amount of rotational restitution across the X axis. The lower, the more restitution occurs.

---

float **angular_limit_x/softness** = `0.5`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

The speed of all rotations across the X axis.

---

float **angular_limit_x/upper_angle** = `0.0`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

The minimum rotation in positive direction to break loose and rotate around the X axis.

---

float **angular_limit_y/damping** = `1.0`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

The amount of rotational damping across the Y axis. The lower, the more damping occurs.

---

bool **angular_limit_y/enabled** = `true`

- void **set_flag_y**(flag: Flag, value: bool)
- bool **get_flag_y**(flag: Flag) const

If `true`, rotation across the Y axis is limited.

---

float **angular_limit_y/erp** = `0.5`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

When rotating across the Y axis, this error tolerance factor defines how much the correction gets slowed down. The lower, the slower.

---

float **angular_limit_y/force_limit** = `0.0`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

The maximum amount of force that can occur, when rotating around the Y axis.

---

float **angular_limit_y/lower_angle** = `0.0`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

The minimum rotation in negative direction to break loose and rotate around the Y axis.

---

float **angular_limit_y/restitution** = `0.0`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

The amount of rotational restitution across the Y axis. The lower, the more restitution occurs.

---

float **angular_limit_y/softness** = `0.5`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

The speed of all rotations across the Y axis.

---

float **angular_limit_y/upper_angle** = `0.0`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

The minimum rotation in positive direction to break loose and rotate around the Y axis.

---

float **angular_limit_z/damping** = `1.0`

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

The amount of rotational damping across the Z axis. The lower, the more damping occurs.

---

bool **angular_limit_z/enabled** = `true`

- void **set_flag_z**(flag: Flag, value: bool)
- bool **get_flag_z**(flag: Flag) const

If `true`, rotation across the Z axis is limited.

---

float **angular_limit_z/erp** = `0.5`

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

When rotating across the Z axis, this error tolerance factor defines how much the correction gets slowed down. The lower, the slower.

---

float **angular_limit_z/force_limit** = `0.0`

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

The maximum amount of force that can occur, when rotating around the Z axis.

---

float **angular_limit_z/lower_angle** = `0.0` 🔗

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

The minimum rotation in negative direction to break loose and rotate around the Z axis.

---

float **angular_limit_z/restitution** = `0.0` 🔗

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

The amount of rotational restitution across the Z axis. The lower, the more restitution occurs.

---

float **angular_limit_z/softness** = `0.5` 🔗

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

The speed of all rotations across the Z axis.

---

float **angular_limit_z/upper_angle** = `0.0` 🔗

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

The minimum rotation in positive direction to break loose and rotate around the Z axis.

---

bool **angular_motor_x/enabled** = `false`

- void **set_flag_x**(flag: Flag, value: bool)
- bool **get_flag_x**(flag: Flag) const

If `true`, a rotating motor at the X axis is enabled.

---

float **angular_motor_x/force_limit** = `300.0`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

Maximum acceleration for the motor at the X axis.

---

float **angular_motor_x/target_velocity** = `0.0`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

Target speed for the motor at the X axis.

---

bool **angular_motor_y/enabled** = `false`

- void **set_flag_y**(flag: Flag, value: bool)
- bool **get_flag_y**(flag: Flag) const

If `true`, a rotating motor at the Y axis is enabled.

---

float **angular_motor_y/force_limit** = `300.0`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

Maximum acceleration for the motor at the Y axis.

float **angular_motor_y/target_velocity** = `0.0`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

Target speed for the motor at the Y axis.

bool **angular_motor_z/enabled** = `false`

- void **set_flag_z**(flag: Flag, value: bool)
- bool **get_flag_z**(flag: Flag) const

If `true`, a rotating motor at the Z axis is enabled.

float **angular_motor_z/force_limit** = `300.0`

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

Maximum acceleration for the motor at the Z axis.

float **angular_motor_z/target_velocity** = `0.0`

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

Target speed for the motor at the Z axis.

float **angular_spring_x/damping** = `0.0`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

bool **angular_spring_x/enabled** = `false` 🔗

- void **set_flag_x**(flag: Flag, value: bool)
- bool **get_flag_x**(flag: Flag) const

There is currently no description for this property. Please help us by contributing one!

---

float **angular_spring_x/equilibrium_point** = `0.0` 🔗

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

float **angular_spring_x/stiffness** = `0.0` 🔗

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

float **angular_spring_y/damping** = `0.0` 🔗

- void **set_param_y**(param: Param, value: float)

- float **get_param_y**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

float **angular_spring_y/damping** = `0.0`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

float **angular_spring_y/equilibrium_point** = `0.0`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

float **angular_spring_y/stiffness** = `0.0`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

float **angular_spring_z/damping** = `0.0`

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

bool **angular_spring_z/enabled** = `false`

- void **set_flag_z**(flag: Flag, value: bool)
- bool **get_flag_z**(flag: Flag) const

There is currently no description for this property. Please help us by contributing one!

---

float **angular_spring_z/equilibrium_point** = `0.0`

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

float **angular_spring_z/stiffness** = `0.0`

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

float **linear_limit_x/damping** = `1.0`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

The amount of damping that happens at the X motion.

---

bool **linear_limit_x/enabled** = `true`

- void **set_flag_x**(flag: Flag, value: bool)
- bool **get_flag_x**(flag: Flag) const

If `true`, the linear motion across the X axis is limited.

---

float **linear_limit_x/lower_distance** = `0.0`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

The minimum difference between the pivot points' X axis.

---

float **linear_limit_x/restitution** = `0.5`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

The amount of restitution on the X axis movement. The lower, the more momentum gets lost.

---

float **linear_limit_x/softness** = `0.7`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

A factor applied to the movement across the X axis. The lower, the slower the movement.

---

float **linear_limit_x/upper_distance** = `0.0`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

The maximum difference between the pivot points' X axis.

---

float **linear_limit_y/damping** = `1.0`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

The amount of damping that happens at the Y motion.

---

bool **linear_limit_y/enabled** = `true`

- void **set_flag_y**(flag: Flag, value: bool)
- bool **get_flag_y**(flag: Flag) const

If `true`, the linear motion across the Y axis is limited.

---

float **linear_limit_y/lower_distance** = `0.0`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

The minimum difference between the pivot points' Y axis.

---

float **linear_limit_y/restitution** = `0.5`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

The amount of restitution on the Y axis movement. The lower, the more momentum gets lost.

---

float **linear_limit_y/softness** = `0.7`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

A factor applied to the movement across the Y axis. The lower, the slower the movement.

---

float **linear_limit_y/upper_distance** = `0.0`

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

The maximum difference between the pivot points' Y axis.

---

float **linear_limit_z/damping** = `1.0`

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

The amount of damping that happens at the Z motion.

---

bool **linear_limit_z/enabled** = `true`

- void **set_flag_z**(flag: Flag, value: bool)
- bool **get_flag_z**(flag: Flag) const

If `true`, the linear motion across the Z axis is limited.

---

float **linear_limit_z/lower_distance** = `0.0`

- void **set_param_z**(param: Param, value: float)

- float **get_param_z**(param: Param) const

The minimum difference between the pivot points' Z axis.

---

float **linear_limit_z/restitution** = `0.5`

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

The amount of restitution on the Z axis movement. The lower, the more momentum gets lost.

---

float **linear_limit_z/softness** = `0.7`

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

A factor applied to the movement across the Z axis. The lower, the slower the movement.

---

float **linear_limit_z/upper_distance** = `0.0`

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

The maximum difference between the pivot points' Z axis.

---

bool **linear_motor_x/enabled** = `false`

- void **set_flag_x**(flag: Flag, value: bool)
- bool **get_flag_x**(flag: Flag) const

If `true`, then there is a linear motor on the X axis. It will attempt to reach the target velocity while staying within the force limits.

---

float **linear_motor_x/force_limit** = `0.0` 🔗

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

The maximum force the linear motor can apply on the X axis while trying to reach the target velocity.

---

float **linear_motor_x/target_velocity** = `0.0` 🔗

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

The speed that the linear motor will attempt to reach on the X axis.

---

bool **linear_motor_y/enabled** = `false` 🔗

- void **set_flag_y**(flag: Flag, value: bool)
- bool **get_flag_y**(flag: Flag) const

If `true`, then there is a linear motor on the Y axis. It will attempt to reach the target velocity while staying within the force limits.

---

float **linear_motor_y/force_limit** = `0.0` 🔗

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

The maximum force the linear motor can apply on the Y axis while trying to reach the target velocity.

---

float **linear_motor_y/target_velocity** = `0.0` 🔗

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

The speed that the linear motor will attempt to reach on the Y axis.

---

bool **linear_motor_z/enabled** = `false` 🔗

- void **set_flag_z**(flag: Flag, value: bool)
- bool **get_flag_z**(flag: Flag) const

If `true`, then there is a linear motor on the Z axis. It will attempt to reach the target velocity while staying within the force limits.

---

float **linear_motor_z/force_limit** = `0.0` 🔗

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

The maximum force the linear motor can apply on the Z axis while trying to reach the target velocity.

---

float **linear_motor_z/target_velocity** = `0.0` 🔗

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

The speed that the linear motor will attempt to reach on the Z axis.

---

float **linear_spring_x/damping** = `0.01` 🔗

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

bool linear_spring_x/enabled = `false`

- void **set_flag_x**(flag: Flag, value: bool)
- bool **get_flag_x**(flag: Flag) const

There is currently no description for this property. Please help us by contributing one!

---

float linear_spring_x/equilibrium_point = `0.0`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

float linear_spring_x/stiffness = `0.01`

- void **set_param_x**(param: Param, value: float)
- float **get_param_x**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

float linear_spring_y/damping = `0.01`

- void **set_param_y**(param: Param, value: float)

- float **get_param_y**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

float **linear_spring_y/enabled** = `false` 🔗

- void **set_flag_y**(flag: Flag, value: bool)
- bool **get_flag_y**(flag: Flag) const

There is currently no description for this property. Please help us by contributing one!

---

float **linear_spring_y/equilibrium_point** = `0.0` 🔗

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

float **linear_spring_y/stiffness** = `0.01` 🔗

- void **set_param_y**(param: Param, value: float)
- float **get_param_y**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

float **linear_spring_z/damping** = `0.01` 🔗

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

bool **linear_spring_z/enabled** = `false`

- void **set_flag_z**(flag: Flag, value: bool)
- bool **get_flag_z**(flag: Flag) const

There is currently no description for this property. Please help us by contributing one!

---

float **linear_spring_z/equilibrium_point** = `0.0`

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

---

float **linear_spring_z/stiffness** = `0.01`

- void **set_param_z**(param: Param, value: float)
- float **get_param_z**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

Method Descriptions

bool **get_flag_x**(flag: Flag) const

There is currently no description for this method. Please help us by contributing one!

---

bool **get_flag_y**(flag: Flag) const

There is currently no description for this method. Please help us by contributing one!

---

bool **get_flag_z**(flag: Flag) const

There is currently no description for this method. Please help us by contributing one!

---

float **get_param_x**(param: Param) const

There is currently no description for this method. Please help us by contributing one!

---

float **get_param_y**(param: Param) const

There is currently no description for this method. Please help us by contributing one!

---

float **get_param_z**(param: Param) const

There is currently no description for this method. Please help us by contributing one!

---

void **set_flag_x**(flag: Flag, value: bool)

There is currently no description for this method. Please help us by contributing one!

void **set_flag_y**(flag: Flag, value: bool)

There is currently no description for this method. Please help us by contributing one!

void **set_flag_z**(flag: Flag, value: bool)

There is currently no description for this method. Please help us by contributing one!

void **set_param_x**(param: Param, value: float)

There is currently no description for this method. Please help us by contributing one!

void **set_param_y**(param: Param, value: float)

There is currently no description for this method. Please help us by contributing one!

void **set_param_z**(param: Param, value: float)

There is currently no description for this method. Please help us by contributing one!

GeometryInstance3D

Inherits: VisualInstance3D **<** Node3D **<** Node **<** Object

Inherited By: CPUParticles3D, CSGShape3D, GPUParticles3D, Label3D, MeshInstance3D, MultiMeshInstance3D, SpriteBase3D

Base node for geometry-based visual instances.

Description

Base node for geometry-based visual instances. Shares some common functionality like visibility and custom materials.

Tutorials

- Visibility ranges (HLOD)

Properties

ShadowCastingSetting	cast_shadow	`1`
AABB	custom_aabb	`AABB(0, 0, 0, 0, 0, 0)`
float	extra_cull_margin	`0.0`

LightmapScale	gi_lightmap_scale	`0`
GIMode	gi_mode	`1`
bool	ignore_occlusion_culling	`false`
float	lod_bias	`1.0`
Material	material_overlay	
Material	material_override	
float	transparency	`0.0`
float	visibility_range_begin	`0.0`
float	visibility_range_begin_margin	`0.0`
float	visibility_range_end	`0.0`
float	visibility_range_end_margin	`0.0`
VisibilityRangeFadeMode	visibility_range_fade_mode	`0`

Methods

Variant	get_instance_shader_parameter(name: StringName) const
void	set_instance_shader_parameter(name: StringName, value: Variant)

Enumerations

enum **ShadowCastingSetting**: 🔗

ShadowCastingSetting **SHADOW_CASTING_SETTING_OFF** = `0`

Will not cast any shadows. Use this to improve performance for small geometry that is unlikely to cast noticeable shadows (such as debris).

ShadowCastingSetting **SHADOW_CASTING_SETTING_ON** = `1`

Will cast shadows from all visible faces in the GeometryInstance3D.

Will take culling into account, so faces not being rendered will not be taken into account when shadow casting.

ShadowCastingSetting **SHADOW_CASTING_SETTING_DOUBLE_SIDED** = `2`

Will cast shadows from all visible faces in the GeometryInstance3D.

Will not take culling into account, so all faces will be taken into account when shadow casting.

ShadowCastingSetting **SHADOW_CASTING_SETTING_SHADOWS_ONLY** = `3`

Will only show the shadows casted from this object.

In other words, the actual mesh will not be visible, only the shadows casted from the mesh will be.

---

enum **GIMode**: 🔗

GIMode **GI_MODE_DISABLED** = `0`

Disabled global illumination mode. Use for dynamic objects that do not contribute to global illumination (such as characters). When using VoxelGI and SDFGI, the geometry will *receive* indirect lighting and reflections but the geometry will not be considered in GI baking.

GIMode **GI_MODE_STATIC** = `1`

Baked global illumination mode. Use for static objects that contribute to global illumination (such as level geometry). This GI mode is effective when using VoxelGI, SDFGI and LightmapGI.

GIMode **GI_MODE_DYNAMIC** = `2`

Dynamic global illumination mode. Use for dynamic objects that contribute to global illumination. This GI mode is only effective when using VoxelGI, but it has a higher performance impact than GI_MODE_STATIC. When using other GI methods, this will act the same as GI_MODE_DISABLED. When using LightmapGI, the object will receive indirect lighting using lightmap probes instead of using the baked lightmap texture.

---

enum **LightmapScale**: 🔗

LightmapScale **LIGHTMAP_SCALE_1X** = `0`

The standard texel density for lightmapping with LightmapGI.

LightmapScale **LIGHTMAP_SCALE_2X** = `1`

Multiplies texel density by 2× for lightmapping with LightmapGI. To ensure consistency in texel density, use this when scaling a mesh by a factor between 1.5 and 3.0.

LightmapScale **LIGHTMAP_SCALE_4X** = `2`

Multiplies texel density by 4× for lightmapping with LightmapGI. To ensure consistency in texel density, use this when scaling a mesh by a factor between 3.0 and 6.0.

LightmapScale **LIGHTMAP_SCALE_8X** = `3`

Multiplies texel density by 8× for lightmapping with LightmapGI. To ensure consistency in texel density, use this when scaling a mesh by a factor greater than 6.0.

LightmapScale **LIGHTMAP_SCALE_MAX** = `4`

Represents the size of the LightmapScale enum.

---

enum **VisibilityRangeFadeMode**: 🔗

VisibilityRangeFadeMode **VISIBILITY_RANGE_FADE_DISABLED** = `0`

Will not fade itself nor its visibility dependencies, hysteresis will be used instead. This is the fastest approach to manual LOD, but it can result in noticeable LOD transitions depending on how the LOD meshes are authored. See visibility_range_begin and Node3D.visibility_parent for more information.

VisibilityRangeFadeMode **VISIBILITY_RANGE_FADE_SELF** = `1`

Will fade-out itself when reaching the limits of its own visibility range. This is slower than VISIBILITY_RANGE_FADE_DISABLED, but it can provide smoother transitions. The fading range is determined by visibility_range_begin_margin and visibility_range_end_margin.

Note: Only supported when using the Forward+ rendering method. When using the Mobile or Compatibility rendering method, this mode acts like VISIBILITY_RANGE_FADE_DISABLED but with hysteresis disabled.

VisibilityRangeFadeMode **VISIBILITY_RANGE_FADE_DEPENDENCIES** = `2`

Will fade-in its visibility dependencies (see Node3D.visibility_parent) when reaching the limits of its own visibility range. This is slower than VISIBILITY_RANGE_FADE_DISABLED, but it can provide smoother transitions. The fading range is determined by visibility_range_begin_margin and visibility_range_end_margin.

Note: Only supported when using the Forward+ rendering method. When using the Mobile or Compatibility rendering method, this mode acts like VISIBILITY_RANGE_FADE_DISABLED but with hysteresis disabled.

Property Descriptions

ShadowCastingSetting **cast_shadow** = `1` 🔗

- void **set_cast_shadows_setting**(value: ShadowCastingSetting)
- ShadowCastingSetting **get_cast_shadows_setting**()

The selected shadow casting flag. See ShadowCastingSetting for possible values.

AABB **custom_aabb** = `AABB(0, 0, 0, 0, 0, 0)` 🔗

- void **set_custom_aabb**(value: AABB)
- AABB **get_custom_aabb**()

Overrides the bounding box of this node with a custom one. This can be used to avoid the expensive AABB recalculation that happens when a skeleton is used with a MeshInstance3D or to have precise control over the MeshInstance3D's bounding box. To use the default AABB, set value to an AABB with all fields set to `0.0`. To avoid frustum culling, set custom_aabb to a very large AABB that covers your entire game world such as `AABB(-10000, -10000, -10000, 20000, 20000, 20000)`. To disable all forms of culling (including occlusion culling), call RenderingServer.instance_set_ignore_culling on the **GeometryInstance3D**'s RID.

---

float **extra_cull_margin** = `0.0` 🔗

- void **set_extra_cull_margin**(value: float)
- float **get_extra_cull_margin**()

The extra distance added to the GeometryInstance3D's bounding box (AABB) to increase its cull box.

LightmapScale **gi_lightmap_scale** = `0` 🔗

- void **set_lightmap_scale**(value: LightmapScale)
- LightmapScale **get_lightmap_scale**()

The texel density to use for lightmapping in LightmapGI. Greater scale values provide higher resolution in the lightmap, which can result in sharper shadows for lights that have both direct and indirect light baked. However, greater scale values will also increase the space taken by the mesh in the lightmap texture, which increases the memory, storage, and bake time requirements. When using a single mesh at different scales, consider adjusting this value to keep the lightmap texel density consistent across meshes.

GIMode **gi_mode** = `1` 🔗

- void **set_gi_mode**(value: GIMode)
- GIMode **get_gi_mode**()

The global illumination mode to use for the whole geometry. To avoid inconsistent results, use a mode that matches the purpose of the mesh during gameplay (static/dynamic).

Note: Lights' bake mode will also affect the global illumination rendering. See Light3D.light_bake_mode.

---

bool **ignore_occlusion_culling** = `false` 🔗

- void **set_ignore_occlusion_culling**(value: bool)
- bool **is_ignoring_occlusion_culling**()

If `true`, disables occlusion culling for this instance. Useful for gizmos that must be rendered even when occlusion culling is in use.

Note: ignore_occlusion_culling does not affect frustum culling (which is what happens when an object is not visible given the camera's angle). To avoid frustum culling, set custom_aabb to a very large AABB that covers your entire game world such as `AABB(-10000, -10000, -10000, 20000, 20000, 20000)`.

---

float **lod_bias** = `1.0` 🔗

- void **set_lod_bias**(value: float)
- float **get_lod_bias**()

Changes how quickly the mesh transitions to a lower level of detail. A value of 0 will force the mesh to its lowest level of detail, a value of 1 will use the default settings, and larger values will keep the mesh in a higher level of detail at farther distances.

Useful for testing level of detail transitions in the editor.

Material **material_overlay** 🔗

- void **set_material_overlay**(value: Material)
- Material **get_material_overlay**()

The material overlay for the whole geometry.

If a material is assigned to this property, it will be rendered on top of any other active material for all the surfaces.

Material **material_override** 🔗

- void **set_material_override**(value: Material)
- Material **get_material_override**()

The material override for the whole geometry.

If a material is assigned to this property, it will be used instead of any material set in any material slot of the mesh.

---

float **transparency** = `0.0` 🔗

- void **set_transparency**(value: float)
- float **get_transparency**()

The transparency applied to the whole geometry (as a multiplier of the materials' existing transparency). `0.0` is fully opaque, while `1.0` is fully transparent. Values greater than `0.0` (exclusive) will force the geometry's materials to go through the transparent pipeline, which is slower to render and can exhibit rendering issues due to incorrect transparency sorting. However, unlike using a transparent material, setting transparency to a value greater than `0.0` (exclusive) will *not* disable shadow rendering.

In spatial shaders, `1.0 - transparency` is set as the default value of the `ALPHA` built-in.

Note: transparency is clamped between `0.0` and `1.0`, so this property cannot be used to make transparent materials more opaque than they originally are.

Note: Only supported when using the Forward+ rendering method. When using the Mobile or Compatibility rendering method, transparency is ignored and is considered as always being `0.0`.

---

float **visibility_range_begin** = `0.0`

- void **set_visibility_range_begin**(value: float)
- float **get_visibility_range_begin**()

Starting distance from which the GeometryInstance3D will be visible, taking visibility_range_begin_margin into account as well. The default value of 0 is used to disable the range check.

---

float **visibility_range_begin_margin** = `0.0`

- void **set_visibility_range_begin_margin**(value: float)
- float **get_visibility_range_begin_margin**()

Margin for the visibility_range_begin threshold. The GeometryInstance3D will only change its visibility state when it goes over or under the visibility_range_begin threshold by this amount.

If visibility_range_fade_mode is VISIBILITY_RANGE_FADE_DISABLED, this acts as a hysteresis distance. If visibility_range_fade_mode is VISIBILITY_RANGE_FADE_SELF or VISIBILITY_RANGE_FADE_DEPENDENCIES, this acts as a fade transition distance and must be set to a value greater than `0.0` for the effect to be noticeable.

---

float **visibility_range_end** = `0.0` 🔗

- void **set_visibility_range_end**(value: float)
- float **get_visibility_range_end**()

Distance from which the GeometryInstance3D will be hidden, taking visibility_range_end_margin into account as well. The default value of 0 is used to disable the range check.

---

float **visibility_range_end_margin** = `0.0` 🔗

- void **set_visibility_range_end_margin**(value: float)
- float **get_visibility_range_end_margin**()

Margin for the visibility_range_end threshold. The GeometryInstance3D will only change its visibility state when it goes over or under the visibility_range_end threshold by this amount.

If visibility_range_fade_mode is VISIBILITY_RANGE_FADE_DISABLED, this acts as a hysteresis distance. If visibility_range_fade_mode is VISIBILITY_RANGE_FADE_SELF or VISIBILITY_RANGE_FADE_DEPENDENCIES, this acts as a fade transition distance and must be set to a value greater than `0.0` for the effect to be noticeable.

VisibilityRangeFadeMode **visibility_range_fade_mode** = `0`

- void **set_visibility_range_fade_mode**(value: VisibilityRangeFadeMode)
- VisibilityRangeFadeMode **get_visibility_range_fade_mode**()

Controls which instances will be faded when approaching the limits of the visibility range. See VisibilityRangeFadeMode for possible values.

Method Descriptions

Variant **get_instance_shader_parameter**(name: StringName) const

Get the value of a shader parameter as set on this instance.

---

void **set_instance_shader_parameter**(name: StringName, value: Variant)

Set the value of a shader uniform for this instance only (per-instance uniform). See also ShaderMaterial.set_shader_parameter to assign a uniform on all instances using the same ShaderMaterial.

Note: For a shader uniform to be assignable on a per-instance basis, it *must* be defined with `instance uniform ...` rather than `uniform ...` in the shader code.

Note: `name` is case-sensitive and must match the name of the uniform in the code exactly (not the capitalized name in the inspector).

Note: Per-instance shader uniforms are currently only available in 3D, so there is no 2D equivalent of this method.

GPUParticles2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A 2D particle emitter.

Description

2D particle node used to create a variety of particle systems and effects. **GPUParticles2D** features an emitter that generates some number of particles at a given rate.

Use the process_material property to add a ParticleProcessMaterial to configure particle appearance and behavior. Alternatively, you can add a ShaderMaterial which will be applied to all particles.

2D particles can optionally collide with LightOccluder2D, but they don't collide with PhysicsBody2D nodes.

Tutorials

- Particle systems (2D)
- 2D Particles Demo [https://godotengine.org/asset-library/asset/2724]
- 2D Dodge The Creeps Demo (uses GPUParticles2D for the trail behind the player) [https://godotengine.org/asset-library/asset/2712]

Properties

int	amount	8

float	amount_ratio	`1.0`
float	collision_base_size	`1.0`
DrawOrder	draw_order	`1`
bool	emitting	`true`
float	explosiveness	`0.0`
int	fixed_fps	`30`
bool	fract_delta	`true`
float	interp_to_end	`0.0`
bool	interpolate	`true`
float	lifetime	`1.0`
bool	local_coords	`false`
bool	one_shot	`false`
float	preprocess	`0.0`

Material	process_material	
float	randomness	`0.0`
float	speed_scale	`1.0`
NodePath	sub_emitter	`NodePath("")`
Texture2D	texture	
bool	trail_enabled	`false`
float	trail_lifetime	`0.3`
int	trail_section_subdivisions	`4`
int	trail_sections	`8`
Rect2	visibility_rect	`Rect2(-100, -100, 200, 200)`

Methods

Rect2	capture_rect() const

void	convert_from_particles(particles: Node)
void	emit_particle(xform: Transform2D, velocity: Vector2, color: Color, custom: Color, flags: int)
void	restart()

Signals

finished() 🔗

Emitted when all active particles have finished processing. To immediately restart the emission cycle, call restart.

Never emitted when one_shot is disabled, as particles will be emitted and processed continuously.

Note: For one_shot emitters, due to the particles being computed on the GPU, there may be a short period after receiving the signal during which setting emitting to `true` will not restart the emission cycle. This delay is avoided by instead calling restart.

Enumerations

enum **DrawOrder**: 🔗

DrawOrder **DRAW_ORDER_INDEX** = `0`

Particles are drawn in the order emitted.

DrawOrder **DRAW_ORDER_LIFETIME** = `1`

Particles are drawn in order of remaining lifetime. In other words, the particle with the highest lifetime is drawn at the front.

DrawOrder **DRAW_ORDER_REVERSE_LIFETIME** = `2`

Particles are drawn in reverse order of remaining lifetime. In other words, the particle with the lowest lifetime is drawn at the front.

---

enum **EmitFlags**: 🔗

EmitFlags **EMIT_FLAG_POSITION** = `1`

Particle starts at the specified position.

EmitFlags **EMIT_FLAG_ROTATION_SCALE** = `2`

Particle starts with specified rotation and scale.

EmitFlags **EMIT_FLAG_VELOCITY** = `4`

Particle starts with the specified velocity vector, which defines the emission direction and speed.

EmitFlags **EMIT_FLAG_COLOR** = `8`

Particle starts with specified color.

EmitFlags **EMIT_FLAG_CUSTOM** = `16`

Particle starts with specified `CUSTOM` data.

Property Descriptions

int **amount** = `8` 🔗

- void **set_amount**(value: int)
- int **get_amount**()

The number of particles to emit in one emission cycle. The effective emission rate is `(amount * amount_ratio) / lifetime` particles per second. Higher values will increase GPU requirements, even if not all particles are visible at a given time or if amount_ratio is decreased.

Note: Changing this value will cause the particle system to restart. To avoid this, change amount_ratio instead.

---

float **amount_ratio** = `1.0` 🔗

- void **set_amount_ratio**(value: float)
- float **get_amount_ratio**()

The ratio of particles that should actually be emitted. If set to a value lower than `1.0`, this will set the amount of emitted particles throughout the lifetime to `amount * amount_ratio`. Unlike changing amount, changing amount_ratio while emitting does not affect already-emitted particles and doesn't cause the particle system to restart. amount_ratio can be used to create effects that make the number of emitted particles vary over time.

Note: Reducing the amount_ratio has no performance benefit, since resources need to be allocated and processed for the total amount of particles regardless of the amount_ratio. If you don't intend to change the number of particles emitted while the particles are emitting, make sure amount_ratio is set to `1` and change amount to your liking instead.

---

float **collision_base_size** = `1.0`

- void **set_collision_base_size**(value: float)
- float **get_collision_base_size**()

Multiplier for particle's collision radius. `1.0` corresponds to the size of the sprite. If particles appear to sink into the ground when colliding, increase this value. If particles appear to float when colliding, decrease this value. Only effective if ParticleProcessMaterial.collision_mode is ParticleProcessMaterial.COLLISION_RIGID or ParticleProcessMaterial.COLLISION_HIDE_ON_CONTACT.

Note: Particles always have a spherical collision shape.

DrawOrder **draw_order** = `1`

- void **set_draw_order**(value: DrawOrder)
- DrawOrder **get_draw_order**()

Particle draw order. Uses DrawOrder values.

---

bool **emitting** = `true`

- void **set_emitting**(value: bool)
- bool **is_emitting**()

If `true`, particles are being emitted. emitting can be used to start and stop particles from emitting. However, if one_shot is `true` setting emitting to `true` will not restart the emission cycle unless all active particles have finished processing. Use the finished signal to be notified once all active particles finish processing.

Note: For one_shot emitters, due to the particles being computed on the GPU, there may be a short period after receiving the finished signal during which setting this to `true` will not restart the emission cycle.

Tip: If your one_shot emitter needs to immediately restart emitting particles once finished signal is received, consider calling restart instead of setting emitting.

---

float **explosiveness** = `0.0`

- void **set_explosiveness_ratio**(value: float)
- float **get_explosiveness_ratio**()

How rapidly particles in an emission cycle are emitted. If greater than `0`, there will be a gap in emissions before the next cycle begins.

---

int **fixed_fps** = `30`

- void **set_fixed_fps**(value: int)
- int **get_fixed_fps**()

The particle system's frame rate is fixed to a value. For example, changing the value to 2 will make the particles render at 2 frames per second. Note this does not slow down the simulation of the particle system itself.

---

bool **fract_delta** = `true`

- void **set_fractional_delta**(value: bool)
- bool **get_fractional_delta**()

If `true`, results in fractional delta calculation which has a smoother particles display effect.

---

float **interp_to_end** = `0.0` 🔗

- void **set_interp_to_end**(value: float)
- float **get_interp_to_end**()

Causes all the particles in this node to interpolate towards the end of their lifetime.

Note: This only works when used with a ParticleProcessMaterial. It needs to be manually implemented for custom process shaders.

---

bool **interpolate** = `true` 🔗

- void **set_interpolate**(value: bool)
- bool **get_interpolate**()

Enables particle interpolation, which makes the particle movement smoother when their fixed_fps is lower than the screen refresh rate.

---

float **lifetime** = `1.0` 🔗

- void **set_lifetime**(value: float)
- float **get_lifetime**()

The amount of time each particle will exist (in seconds). The effective emission rate is `(amount * amount_ratio) / lifetime` particles per second.

---

bool **local_coords** = `false` 🔗

- void **set_use_local_coordinates**(value: bool)
- bool **get_use_local_coordinates**()

If `true`, particles use the parent node's coordinate space (known as local coordinates). This will cause particles to move and rotate along the **GPUParticles2D** node (and its parents) when it is moved or rotated. If `false`, particles use global coordinates; they will not move or rotate along the **GPUParticles2D** node (and its parents) when it is moved or rotated.

---

bool **one_shot** = `false` 🔗

- void **set_one_shot**(value: bool)
- bool **get_one_shot**()

If `true`, only one emission cycle occurs. If set `true` during a cycle, emission will stop at the cycle's end.

---

float **preprocess** = `0.0` 🔗

- void **set_pre_process_time**(value: float)
- float **get_pre_process_time**()

Particle system starts as if it had already run for this many seconds.

Material **process_material** 🔗

- void **set_process_material**(value: Material)
- Material **get_process_material**()

Material for processing particles. Can be a ParticleProcessMaterial or a ShaderMaterial.

---

float **randomness** = `0.0` 🔗

- void **set_randomness_ratio**(value: float)

- float **get_randomness_ratio**()

Emission lifetime randomness ratio.

---

float **speed_scale** = `1.0`

- void **set_speed_scale**(value: float)
- float **get_speed_scale**()

Particle system's running speed scaling ratio. A value of `0` can be used to pause the particles.

NodePath **sub_emitter** = `NodePath("")`

- void **set_sub_emitter**(value: NodePath)
- NodePath **get_sub_emitter**()

Path to another **GPUParticles2D** node that will be used as a subemitter (see ParticleProcessMaterial.sub_emitter_mode). Subemitters can be used to achieve effects such as fireworks, sparks on collision, bubbles popping into water drops, and more.

Note: When sub_emitter is set, the target **GPUParticles2D** node will no longer emit particles on its own.

Texture2D **texture**

- void **set_texture**(value: Texture2D)
- Texture2D **get_texture**()

Particle texture. If `null`, particles will be squares with a size of 1×1 pixels.

Note: To use a flipbook texture, assign a new CanvasItemMaterial to the **GPUParticles2D**'s CanvasItem.material property, then enable CanvasItemMaterial.particles_animation and set CanvasItemMaterial.particles_anim_h_frames, CanvasItemMaterial.particles_anim_v_frames, and CanvasItemMaterial.particles_anim_loop to match the flipbook texture.

---

bool **trail_enabled** = `false`

- void **set_trail_enabled**(value: bool)
- bool **is_trail_enabled**()

If `true`, enables particle trails using a mesh skinning system.

Note: Unlike GPUParticles3D, the number of trail sections and subdivisions is set with the trail_sections and trail_section_subdivisions properties.

---

float **trail_lifetime** = `0.3`

- void **set_trail_lifetime**(value: float)
- float **get_trail_lifetime**()

The amount of time the particle's trail should represent (in seconds). Only effective if trail_enabled is `true`.

---

int **trail_section_subdivisions** = `4`

- void **set_trail_section_subdivisions**(value: int)
- int **get_trail_section_subdivisions**()

The number of subdivisions to use for the particle trail rendering. Higher values can result in smoother trail curves, at the cost of performance due to increased mesh complexity. See also trail_sections. Only effective if trail_enabled is `true`.

---

int **trail_sections** = `8` 🔗

- void **set_trail_sections**(value: int)
- int **get_trail_sections**()

The number of sections to use for the particle trail rendering. Higher values can result in smoother trail curves, at the cost of performance due to increased mesh complexity. See also trail_section_subdivisions. Only effective if trail_enabled is `true`.

Rect2 **visibility_rect** = `Rect2(-100, -100, 200, 200)` 🔗

- void **set_visibility_rect**(value: Rect2)
- Rect2 **get_visibility_rect**()

The Rect2 that determines the node's region which needs to be visible on screen for the particle system to be active.

Grow the rect if particles suddenly appear/disappear when the node enters/exits the screen. The Rect2 can be grown via code or with the **Particles → Generate Visibility Rect** editor tool.

Method Descriptions

Rect2 **capture_rect**() const 🔗

Returns a rectangle containing the positions of all existing particles.

Note: When using threaded rendering this method synchronizes the rendering thread. Calling it often may have a negative impact on performance.

void **convert_from_particles**(particles: Node)

Sets this node's properties to match a given CPUParticles2D node.

void **emit_particle**(xform: Transform2D, velocity: Vector2, color: Color, custom: Color, flags: int)

Emits a single particle. Whether `xform`, `velocity`, `color` and `custom` are applied depends on the value of `flags`. See EmitFlags.

The default ParticleProcessMaterial will overwrite `color` and use the contents of `custom` as `(rotation, age, animation, lifetime)`.

void **restart**()

Restarts the particle emission cycle, clearing existing particles. To avoid particles vanishing from the viewport, wait for the finished signal before calling.

Note: The finished signal is only emitted by one_shot emitters.

GPUParticles3D

Inherits: GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A 3D particle emitter.

Description

3D particle node used to create a variety of particle systems and effects. **GPUParticles3D** features an emitter that generates some number of particles at a given rate.

Use process_material to add a ParticleProcessMaterial to configure particle appearance and behavior. Alternatively, you can add a ShaderMaterial which will be applied to all particles.

Tutorials

- Particle systems (3D)
- Controlling thousands of fish with Particles
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

int	amount	`8`
float	amount_ratio	`1.0`

float	collision_base_size	`0.01`
DrawOrder	draw_order	`0`
Mesh	draw_pass_1	
Mesh	draw_pass_2	
Mesh	draw_pass_3	
Mesh	draw_pass_4	
int	draw_passes	`1`
Skin	draw_skin	
bool	emitting	`true`
float	explosiveness	`0.0`
int	fixed_fps	`30`
bool	fract_delta	`true`

float	interp_to_end	`0.0`
bool	interpolate	`true`
float	lifetime	`1.0`
bool	local_coords	`false`
bool	one_shot	`false`
float	preprocess	`0.0`
Material	process_material	
float	randomness	`0.0`
float	speed_scale	`1.0`
NodePath	sub_emitter	`NodePath("")`
bool	trail_enabled	`false`
float	trail_lifetime	`0.3`
TransformAlign	transform_align	`0`

AABB	visibility_aabb	`AABB(-4, -4, -4, 8, 8, 8)`

Methods

AABB	capture_aabb() const
void	convert_from_particles(particles: Node)
void	emit_particle(xform: Transform3D, velocity: Vector3, color: Color, custom: Color, flags: int)
Mesh	get_draw_pass_mesh(pass: int) const
void	restart()
void	set_draw_pass_mesh(pass: int, mesh: Mesh)

Signals

finished() 🔗

Emitted when all active particles have finished processing. To immediately emit new particles, call restart.

Never emitted when one_shot is disabled, as particles will be emitted and processed continuously.

Note: For one_shot emitters, due to the particles being computed on the GPU, there may be a short period after receiving the signal during which setting emitting to `true` will not restart the emission cycle. This delay is avoided by instead calling restart.

Enumerations

enum **DrawOrder**: 🔗

DrawOrder **DRAW_ORDER_INDEX** = `0`

Particles are drawn in the order emitted.

DrawOrder **DRAW_ORDER_LIFETIME** = `1`

Particles are drawn in order of remaining lifetime. In other words, the particle with the highest lifetime is drawn at the front.

DrawOrder **DRAW_ORDER_REVERSE_LIFETIME** = `2`

Particles are drawn in reverse order of remaining lifetime. In other words, the particle with the lowest lifetime is drawn at the front.

DrawOrder **DRAW_ORDER_VIEW_DEPTH** = `3`

Particles are drawn in order of depth.

---

enum **EmitFlags**: 🔗

EmitFlags **EMIT_FLAG_POSITION** = `1`

Particle starts at the specified position.

EmitFlags **EMIT_FLAG_ROTATION_SCALE** = `2`

Particle starts with specified rotation and scale.

EmitFlags **EMIT_FLAG_VELOCITY** = `4`

Particle starts with the specified velocity vector, which defines the emission direction and speed.

EmitFlags **EMIT_FLAG_COLOR** = `8`

Particle starts with specified color.

EmitFlags **EMIT_FLAG_CUSTOM** = `16`

Particle starts with specified `CUSTOM` data.

---

enum **TransformAlign**: 🔗

TransformAlign **TRANSFORM_ALIGN_DISABLED** = `0`

There is currently no description for this enum. Please help us by contributing one!

TransformAlign **TRANSFORM_ALIGN_Z_BILLBOARD** = `1`

There is currently no description for this enum. Please help us by contributing one!

TransformAlign **TRANSFORM_ALIGN_Y_TO_VELOCITY** = `2`

There is currently no description for this enum. Please help us by contributing one!

TransformAlign
TRANSFORM_ALIGN_Z_BILLBOARD_Y_TO_VELOCITY = `3`

There is currently no description for this enum. Please help us by contributing one!

Constants

MAX_DRAW_PASSES = `4`

Maximum number of draw passes supported.

Property Descriptions

int **amount** = `8`

- void **set_amount**(value: int)
- int **get_amount**()

The number of particles to emit in one emission cycle. The effective emission rate is `(amount * amount_ratio) / lifetime` particles per second. Higher values will increase GPU requirements, even if not all particles are visible at a given time or if amount_ratio is decreased.

Note: Changing this value will cause the particle system to restart. To avoid this, change amount_ratio instead.

---

float **amount_ratio** = `1.0`

- void **set_amount_ratio**(value: float)
- float **get_amount_ratio**()

The ratio of particles that should actually be emitted. If set to a value lower than `1.0`, this will set the amount of emitted particles throughout the lifetime to `amount * amount_ratio`. Unlike changing amount, changing amount_ratio while emitting does not affect already-emitted particles and doesn't cause the particle system to restart. amount_ratio can be used to create effects that make the number of emitted particles vary over time.

Note: Reducing the amount_ratio has no performance benefit, since resources need to be allocated and processed for the total amount of particles regardless of the amount_ratio. If you don't intend to change the number of particles emitted while the particles are emitting, make sure amount_ratio is set to `1` and change amount to your liking instead.

---

float **collision_base_size** = `0.01` 🔗

- void **set_collision_base_size**(value: float)
- float **get_collision_base_size**()

The base diameter for particle collision in meters. If particles appear to sink into the ground when colliding, increase this value. If particles appear to float when colliding, decrease this value. Only effective if ParticleProcessMaterial.collision_mode is ParticleProcessMaterial.COLLISION_RIGID or ParticleProcessMaterial.COLLISION_HIDE_ON_CONTACT.

Note: Particles always have a spherical collision shape.

DrawOrder **draw_order** = `0` 🔗

- void **set_draw_order**(value: DrawOrder)
- DrawOrder **get_draw_order**()

Particle draw order. Uses DrawOrder values.

Note: DRAW_ORDER_INDEX is the only option that supports motion vectors for effects like TAA. It is suggested to use this draw order if the particles are opaque to fix ghosting artifacts.

Mesh **draw_pass_1** 🔗

- void **set_draw_pass_mesh**(pass: int, mesh: Mesh)
- Mesh **get_draw_pass_mesh**(pass: int) const

Mesh that is drawn for the first draw pass.

Mesh **draw_pass_2** 🔗

- void **set_draw_pass_mesh**(pass: int, mesh: Mesh)
- Mesh **get_draw_pass_mesh**(pass: int) const

Mesh that is drawn for the second draw pass.

Mesh **draw_pass_3** 🔗

- void **set_draw_pass_mesh**(pass: int, mesh: Mesh)
- Mesh **get_draw_pass_mesh**(pass: int) const

Mesh that is drawn for the third draw pass.

Mesh **draw_pass_4** 🔗

- void **set_draw_pass_mesh**(pass: int, mesh: Mesh)
- Mesh **get_draw_pass_mesh**(pass: int) const

Mesh that is drawn for the fourth draw pass.

---

int **draw_passes** = `1` 🔗

- void **set_draw_passes**(value: int)
- int **get_draw_passes**()

The number of draw passes when rendering particles.

Skin **draw_skin** 🔗

- void **set_skin**(value: Skin)
- Skin **get_skin**()

There is currently no description for this property. Please help us by contributing one!

---

bool **emitting** = `true` 🔗

- void **set_emitting**(value: bool)
- bool **is_emitting**()

If `true`, particles are being emitted. emitting can be used to start and stop particles from emitting. However, if one_shot is `true` setting emitting to `true` will not restart the emission cycle unless all active particles have finished processing. Use the finished signal to be notified once all active particles finish processing.

Note: For one_shot emitters, due to the particles being computed on the GPU, there may be a short period after receiving the finished signal during which setting this to `true` will not restart the emission cycle.

Tip: If your one_shot emitter needs to immediately restart emitting particles once finished signal is received, consider calling restart instead of setting emitting.

---

float **explosiveness** = `0.0` 🔗

- void **set_explosiveness_ratio**(value: float)
- float **get_explosiveness_ratio**()

Time ratio between each emission. If `0`, particles are emitted continuously. If `1`, all particles are emitted simultaneously.

---

int **fixed_fps** = `30` 🔗

- void **set_fixed_fps**(value: int)
- int **get_fixed_fps**()

The particle system's frame rate is fixed to a value. For example, changing the value to 2 will make the particles render at 2 frames per second. Note this does not slow down the simulation of the particle system itself.

---

bool **fract_delta** = `true` 🔗

- void **set_fractional_delta**(value: bool)
- bool **get_fractional_delta**()

If `true`, results in fractional delta calculation which has a smoother particles display effect.

---

float **interp_to_end** = `0.0` 🔗

- void **set_interp_to_end**(value: float)
- float **get_interp_to_end**()

Causes all the particles in this node to interpolate towards the end of their lifetime.

Note: This only works when used with a ParticleProcessMaterial. It needs to be manually implemented for custom process shaders.

---

bool **interpolate** = `true` 🔗

- void **set_interpolate**(value: bool)
- bool **get_interpolate**()

Enables particle interpolation, which makes the particle movement smoother when their fixed_fps is lower than the screen refresh rate.

---

float **lifetime** = `1.0` 🔗

- void **set_lifetime**(value: float)
- float **get_lifetime**()

The amount of time each particle will exist (in seconds). The effective emission rate is `(amount * amount_ratio) / lifetime` particles per second.

---

bool **local_coords** = `false` 🔗

- void **set_use_local_coordinates**(value: bool)
- bool **get_use_local_coordinates**()

If `true`, particles use the parent node's coordinate space (known as local coordinates). This will cause particles to move and rotate along the **GPUParticles3D** node (and its parents) when it is moved or rotated. If `false`, particles use global coordinates; they will not move or rotate along the **GPUParticles3D** node (and its parents) when it is moved or rotated.

---

bool **one_shot** = `false` 🔗

- void **set_one_shot**(value: bool)
- bool **get_one_shot**()

If `true`, only the number of particles equal to amount will be emitted.

---

float **preprocess** = `0.0`

- void **set_pre_process_time**(value: float)
- float **get_pre_process_time**()

Amount of time to preprocess the particles before animation starts. Lets you start the animation some time after particles have started emitting.

Material **process_material**

- void **set_process_material**(value: Material)
- Material **get_process_material**()

Material for processing particles. Can be a ParticleProcessMaterial or a ShaderMaterial.

---

float **randomness** = `0.0`

- void **set_randomness_ratio**(value: float)
- float **get_randomness_ratio**()

Emission randomness ratio.

---

float **speed_scale** = `1.0`

- void **set_speed_scale**(value: float)

- float **get_speed_scale**()

Speed scaling ratio. A value of `0` can be used to pause the particles.

NodePath **sub_emitter** = `NodePath("")` 🔗

- void **set_sub_emitter**(value: NodePath)
- NodePath **get_sub_emitter**()

Path to another **GPUParticles3D** node that will be used as a subemitter (see ParticleProcessMaterial.sub_emitter_mode). Subemitters can be used to achieve effects such as fireworks, sparks on collision, bubbles popping into water drops, and more.

Note: When sub_emitter is set, the target **GPUParticles3D** node will no longer emit particles on its own.

---

bool **trail_enabled** = `false` 🔗

- void **set_trail_enabled**(value: bool)
- bool **is_trail_enabled**()

If `true`, enables particle trails using a mesh skinning system. Designed to work with RibbonTrailMesh and TubeTrailMesh.

Note: BaseMaterial3D.use_particle_trails must also be enabled on the particle mesh's material. Otherwise, setting trail_enabled to `true` will have no effect.

Note: Unlike GPUParticles2D, the number of trail sections and subdivisions is set in the RibbonTrailMesh or the TubeTrailMesh's properties.

---

float **trail_lifetime** = `0.3` 🔗

- void **set_trail_lifetime**(value: float)
- float **get_trail_lifetime**()

The amount of time the particle's trail should represent (in seconds). Only effective if trail_enabled is `true`.

TransformAlign **transform_align** = `0`

- void **set_transform_align**(value: TransformAlign)
- TransformAlign **get_transform_align**()

There is currently no description for this property. Please help us by contributing one!

AABB **visibility_aabb** = `AABB(-4, -4, -4, 8, 8, 8)`

- void **set_visibility_aabb**(value: AABB)
- AABB **get_visibility_aabb**()

The AABB that determines the node's region which needs to be visible on screen for the particle system to be active. GeometryInstance3D.extra_cull_margin is added on each of the AABB's axes. Particle collisions and attraction will only occur within this area.

Grow the box if particles suddenly appear/disappear when the node enters/exits the screen. The AABB can be grown via code or with the **Particles → Generate AABB** editor tool.

Note: visibility_aabb is overridden by GeometryInstance3D.custom_aabb if that property is set to a non-default value.

Method Descriptions

AABB **capture_aabb**() const 🔗

Returns the axis-aligned bounding box that contains all the particles that are active in the current frame.

---

void **convert_from_particles**(particles: Node) 🔗

Sets this node's properties to match a given CPUParticles3D node.

---

void **emit_particle**(xform: Transform3D, velocity: Vector3, color: Color, custom: Color, flags: int) 🔗

Emits a single particle. Whether `xform`, `velocity`, `color` and `custom` are applied depends on the value of `flags`. See EmitFlags.

The default ParticleProcessMaterial will overwrite `color` and use the contents of `custom` as `(rotation, age, animation, lifetime)`.

Mesh **get_draw_pass_mesh**(pass: int) const 🔗

Returns the Mesh that is drawn at index `pass`.

---

void **restart**() 🔗

Restarts the particle emission cycle, clearing existing particles. To avoid particles vanishing from the viewport, wait for the finished signal before calling.

Note: The finished signal is only emitted by one_shot emitters.

---

void **set_draw_pass_mesh**(pass: int, mesh: Mesh) 🔗

Sets the Mesh that is drawn at index `pass`.

GPUParticlesAttractor3D

Inherits: VisualInstance3D **<** Node3D **<** Node **<** Object

Inherited By: GPUParticlesAttractorBox3D, GPUParticlesAttractorSphere3D, GPUParticlesAttractorVectorField3D

Abstract base class for 3D particle attractors.

Description

Particle attractors can be used to attract particles towards the attractor's origin, or to push them away from the attractor's origin.

Particle attractors work in real-time and can be moved, rotated and scaled during gameplay. Unlike collision shapes, non-uniform scaling of attractors is also supported.

Attractors can be temporarily disabled by hiding them, or by setting their strength to `0.0`.

Note: Particle attractors only affect GPUParticles3D, not CPUParticles3D.

Properties

float	attenuation	`1.0`
int	cull_mask	`4294967295`

float	directionality	`0.0`
float	strength	`1.0`

Property Descriptions

float **attenuation** = `1.0`

- void **set_attenuation**(value: float)
- float **get_attenuation**()

The particle attractor's attenuation. Higher values result in more gradual pushing of particles as they come closer to the attractor's origin. Zero or negative values will cause particles to be pushed very fast as soon as the touch the attractor's edges.

int **cull_mask** = `4294967295`

- void **set_cull_mask**(value: int)
- int **get_cull_mask**()

The particle rendering layers (VisualInstance3D.layers) that will be affected by the attractor. By default, all particles are affected by an attractor.

After configuring particle nodes accordingly, specific layers can be unchecked to prevent certain particles from being affected by attractors. For example, this can be used if you're using an attractor as part of a spell effect but don't want the attractor to affect unrelated weather particles at the same position.

Particle attraction can also be disabled on a per-process material basis by setting ParticleProcessMaterial.attractor_interaction_enabled on the GPUParticles3D node.

---

float **directionality** = `0.0`

- void **set_directionality**(value: float)
- float **get_directionality**()

Adjusts how directional the attractor is. At `0.0`, the attractor is not directional at all: it will attract particles towards its center. At `1.0`, the attractor is fully directional: particles will always be pushed towards local -Z (or +Z if strength is negative).

Note: If directionality is greater than `0.0`, the direction in which particles are pushed can be changed by rotating the **GPUParticlesAttractor3D** node.

---

float **strength** = `1.0`

- void **set_strength**(value: float)
- float **get_strength**()

Adjusts the strength of the attractor. If strength is negative, particles will be pushed in the opposite direction. Particles will be pushed *away* from the attractor's origin if directionality is `0.0`, or towards local +Z if directionality is greater than `0.0`.

GPUParticlesAttractorBox3D

Inherits: GPUParticlesAttractor3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A box-shaped attractor that influences particles from GPUParticles3D nodes.

Description

A box-shaped attractor that influences particles from GPUParticles3D nodes. Can be used to attract particles towards its origin, or to push them away from its origin.

Particle attractors work in real-time and can be moved, rotated and scaled during gameplay. Unlike collision shapes, non-uniform scaling of attractors is also supported.

Note: Particle attractors only affect GPUParticles3D, not CPUParticles3D.

Properties

Vector3	size	`Vector3(2, 2, 2)`

Property Descriptions

Vector3 **size** = `Vector3(2, 2, 2)`

- void **set_size**(value: Vector3)
- Vector3 **get_size**()

The attractor box's size in 3D units.

GPUParticlesAttractorSphere3D

Inherits: GPUParticlesAttractor3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A spheroid-shaped attractor that influences particles from GPUParticles3D nodes.

Description

A spheroid-shaped attractor that influences particles from GPUParticles3D nodes. Can be used to attract particles towards its origin, or to push them away from its origin.

Particle attractors work in real-time and can be moved, rotated and scaled during gameplay. Unlike collision shapes, non-uniform scaling of attractors is also supported.

Note: Particle attractors only affect GPUParticles3D, not CPUParticles3D.

Properties

float	radius	`1.0`

Property Descriptions

float **radius** = `1.0` 🔗

- void **set_radius**(value: float)
- float **get_radius**()

The attractor sphere's radius in 3D units.

Note: Stretched ellipses can be obtained by using non-uniform scaling on the **GPUParticlesAttractorSphere3D** node.

GPUParticlesAttractorVectorField3D

Inherits: GPUParticlesAttractor3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A box-shaped attractor with varying directions and strengths defined in it that influences particles from GPUParticles3D nodes.

Description

A box-shaped attractor with varying directions and strengths defined in it that influences particles from GPUParticles3D nodes.

Unlike GPUParticlesAttractorBox3D, **GPUParticlesAttractorVectorField3D** uses a texture to affect attraction strength within the box. This can be used to create complex attraction scenarios where particles travel in different directions depending on their location. This can be useful for weather effects such as sandstorms.

Particle attractors work in real-time and can be moved, rotated and scaled during gameplay. Unlike collision shapes, non-uniform scaling of attractors is also supported.

Note: Particle attractors only affect GPUParticles3D, not CPUParticles3D.

Properties

Vector3	size	`Vector3(2, 2, 2)`
Texture3D	texture	

Property Descriptions

Vector3 **size** = `Vector3(2, 2, 2)` 🔗

- void **set_size**(value: Vector3)
- Vector3 **get_size**()

The size of the vector field box in 3D units.

Texture3D **texture** 🔗

- void **set_texture**(value: Texture3D)
- Texture3D **get_texture**()

The 3D texture to be used. Values are linearly interpolated between the texture's pixels.

Note: To get better performance, the 3D texture's resolution should reflect the size of the attractor. Since particle attraction is usually low-frequency data, the texture can be kept at a low resolution such as 64×64×64.

GPUParticlesCollision3D

Inherits: VisualInstance3D **<** Node3D **<** Node **<** Object

Inherited By: GPUParticlesCollisionBox3D, GPUParticlesCollisionHeightField3D, GPUParticlesCollisionSDF3D, GPUParticlesCollisionSphere3D

Abstract base class for 3D particle collision shapes affecting GPUParticles3D nodes.

Description

Particle collision shapes can be used to make particles stop or bounce against them.

Particle collision shapes work in real-time and can be moved, rotated and scaled during gameplay. Unlike attractors, non-uniform scaling of collision shapes is *not* supported.

Particle collision shapes can be temporarily disabled by hiding them.

Note: ParticleProcessMaterial.collision_mode must be ParticleProcessMaterial.COLLISION_RIGID or ParticleProcessMaterial.COLLISION_HIDE_ON_CONTACT on the GPUParticles3D's process material for collision to work.

Note: Particle collision only affects GPUParticles3D, not CPUParticles3D.

Note: Particles pushed by a collider that is being moved will not be interpolated, which can result in visible stuttering. This can be alleviated by setting GPUParticles3D.fixed_fps to `0` or a value that matches or exceeds the target framerate.

Properties

int	cull_mask	`4294967295`

Property Descriptions

int **cull_mask** = `4294967295` 🔗

- void **set_cull_mask**(value: int)
- int **get_cull_mask**()

The particle rendering layers (VisualInstance3D.layers) that will be affected by the collision shape. By default, all particles that have ParticleProcessMaterial.collision_mode set to ParticleProcessMaterial.COLLISION_RIGID or ParticleProcessMaterial.COLLISION_HIDE_ON_CONTACT will be affected by a collision shape.

After configuring particle nodes accordingly, specific layers can be unchecked to prevent certain particles from being affected by attractors. For example, this can be used if you're using an attractor as part of a spell effect but don't want the attractor to affect unrelated weather particles at the same position.

Particle attraction can also be disabled on a per-process material basis by setting ParticleProcessMaterial.attractor_interaction_enabled on the GPUParticles3D node.

GPUParticlesCollisionBox3D

Inherits: GPUParticlesCollision3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A box-shaped 3D particle collision shape affecting GPUParticles3D nodes.

Description

A box-shaped 3D particle collision shape affecting GPUParticles3D nodes.

Particle collision shapes work in real-time and can be moved, rotated and scaled during gameplay. Unlike attractors, non-uniform scaling of collision shapes is *not* supported.

Note: ParticleProcessMaterial.collision_mode must be ParticleProcessMaterial.COLLISION_RIGID or ParticleProcessMaterial.COLLISION_HIDE_ON_CONTACT on the GPUParticles3D's process material for collision to work.

Note: Particle collision only affects GPUParticles3D, not CPUParticles3D.

Properties

Vector3	size	`Vector3(2, 2, 2)`

Property Descriptions

Vector3 **size** = `Vector3(2, 2, 2)` 🔗

- void **set_size**(value: Vector3)
- Vector3 **get_size**()

The collision box's size in 3D units.

GPUParticlesCollisionHeightField3D

Inherits: GPUParticlesCollision3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A real-time heightmap-shaped 3D particle collision shape affecting GPUParticles3D nodes.

Description

A real-time heightmap-shaped 3D particle collision shape affecting GPUParticles3D nodes.

Heightmap shapes allow for efficiently representing collisions for convex and concave objects with a single "floor" (such as terrain). This is less flexible than GPUParticlesCollisionSDF3D, but it doesn't require a baking step.

GPUParticlesCollisionHeightField3D can also be regenerated in real-time when it is moved, when the camera moves, or even continuously. This makes **GPUParticlesCollisionHeightField3D** a good choice for weather effects such as rain and snow and games with highly dynamic geometry. However, this class is limited since heightmaps cannot represent overhangs (e.g. indoors or caves).

Note: ParticleProcessMaterial.collision_mode must be `true` on the GPUParticles3D's process material for collision to work.

Note: Particle collision only affects GPUParticles3D, not CPUParticles3D.

Properties

bool	follow_camera_enabled	`false`
Resolution	resolution	`2`
Vector3	size	`Vector3(2, 2, 2)`
UpdateMode	update_mode	`0`

Enumerations

enum **Resolution**: 🔗

Resolution **RESOLUTION_256** = `0`

Generate a 256×256 heightmap. Intended for small-scale scenes, or larger scenes with no distant particles.

Resolution **RESOLUTION_512** = `1`

Generate a 512×512 heightmap. Intended for medium-scale scenes, or larger scenes with no distant particles.

Resolution **RESOLUTION_1024** = `2`

Generate a 1024×1024 heightmap. Intended for large scenes with distant particles.

Resolution **RESOLUTION_2048** = `3`

Generate a 2048×2048 heightmap. Intended for very large scenes with distant particles.

Resolution **RESOLUTION_4096** = `4`

Generate a 4096×4096 heightmap. Intended for huge scenes with distant particles.

Resolution **RESOLUTION_8192** = `5`

Generate a 8192×8192 heightmap. Intended for gigantic scenes with distant particles.

Resolution **RESOLUTION_MAX** = `6`

Represents the size of the Resolution enum.

---

enum **UpdateMode**: 🔗

UpdateMode **UPDATE_MODE_WHEN_MOVED** = `0`

Only update the heightmap when the **GPUParticlesCollisionHeightField3D** node is moved, or when the camera moves if follow_camera_enabled is `true`. An update can be forced by slightly moving the **GPUParticlesCollisionHeightField3D** in any direction, or by calling RenderingServer.particles_collision_height_field_update.

UpdateMode **UPDATE_MODE_ALWAYS** = `1`

Update the heightmap every frame. This has a significant performance cost. This update should only be used when geometry that particles can collide with changes significantly during gameplay.

Property Descriptions

bool **follow_camera_enabled** = `false`

- void **set_follow_camera_enabled**(value: bool)
- bool **is_follow_camera_enabled**()

If `true`, the **GPUParticlesCollisionHeightField3D** will follow the current camera in global space. The **GPUParticlesCollisionHeightField3D** does not need to be a child of the Camera3D node for this to work.

Following the camera has a performance cost, as it will force the heightmap to update whenever the camera moves. Consider lowering resolution to improve performance if follow_camera_enabled is `true`.

Resolution **resolution** = `2`

- void **set_resolution**(value: Resolution)
- Resolution **get_resolution**()

Higher resolutions can represent small details more accurately in large scenes, at the cost of lower performance. If update_mode is UPDATE_MODE_ALWAYS, consider using the lowest resolution possible.

Vector3 **size** = `Vector3(2, 2, 2)`

- void **set_size**(value: Vector3)
- Vector3 **get_size**()

The collision heightmap's size in 3D units. To improve heightmap quality, size should be set as small as possible while covering the parts of the scene you need.

UpdateMode **update_mode** = `0`

- void **set_update_mode**(value: UpdateMode)
- UpdateMode **get_update_mode**()

The update policy to use for the generated heightmap.

GPUParticlesCollisionSDF3D

Inherits: GPUParticlesCollision3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A baked signed distance field 3D particle collision shape affecting GPUParticles3D nodes.

Description

A baked signed distance field 3D particle collision shape affecting GPUParticles3D nodes.

Signed distance fields (SDF) allow for efficiently representing approximate collision shapes for convex and concave objects of any shape. This is more flexible than GPUParticlesCollisionHeightField3D, but it requires a baking step.

Baking: The signed distance field texture can be baked by selecting the **GPUParticlesCollisionSDF3D** node in the editor, then clicking **Bake SDF** at the top of the 3D viewport. Any *visible* MeshInstance3Ds within the size will be taken into account for baking, regardless of their GeometryInstance3D.gi_mode.

Note: Baking a **GPUParticlesCollisionSDF3D**'s texture is only possible within the editor, as there is no bake method exposed for use in exported projects. However, it's still possible to load pre-baked Texture3Ds into its texture property in an exported project.

Note: ParticleProcessMaterial.collision_mode must be ParticleProcessMaterial.COLLISION_RIGID or ParticleProcessMaterial.COLLISION_HIDE_ON_CONTACT on the GPUParticles3D's process material for collision to work.

Note: Particle collision only affects GPUParticles3D, not CPUParticles3D.

Properties

int	bake_mask	`4294967295`
Resolution	resolution	`2`
Vector3	size	`Vector3(2, 2, 2)`
Texture3D	texture	
float	thickness	`1.0`

Methods

bool	get_bake_mask_value(layer_number: int) const
void	set_bake_mask_value(layer_number: int, value: bool)

Enumerations

enum **Resolution**: 🔗

Resolution **RESOLUTION_16** = `0`

Bake a 16×16×16 signed distance field. This is the fastest option, but also the least precise.

Resolution **RESOLUTION_32** = `1`

Bake a 32×32×32 signed distance field.

Resolution **RESOLUTION_64** = `2`

Bake a 64×64×64 signed distance field.

Resolution **RESOLUTION_128** = `3`

Bake a 128×128×128 signed distance field.

Resolution **RESOLUTION_256** = `4`

Bake a 256×256×256 signed distance field.

Resolution **RESOLUTION_512** = `5`

Bake a 512×512×512 signed distance field. This is the slowest option, but also the most precise.

Resolution **RESOLUTION_MAX** = `6`

Represents the size of the Resolution enum.

Property Descriptions

int **bake_mask** = `4294967295` 🔗

- void **set_bake_mask**(value: int)
- int **get_bake_mask**()

The visual layers to account for when baking the particle collision SDF. Only MeshInstance3Ds whose VisualInstance3D.layers match with this bake_mask will be included in the generated particle collision SDF. By default, all objects are taken into account for the particle collision SDF baking.

Resolution **resolution** = `2` 🔗

- void **set_resolution**(value: Resolution)
- Resolution **get_resolution**()

The bake resolution to use for the signed distance field texture. The texture must be baked again for changes to the resolution property to be effective. Higher resolutions have a greater performance cost and take more time to bake. Higher resolutions also result in larger baked textures, leading to increased VRAM and storage space requirements. To improve performance and reduce bake times, use the lowest resolution possible for the object you're representing the collision of.

Vector3 **size** = `Vector3(2, 2, 2)` 🔗

- void **set_size**(value: Vector3)
- Vector3 **get_size**()

The collision SDF's size in 3D units. To improve SDF quality, the size should be set as small as possible while covering the parts of the scene you need.

Texture3D **texture** 🔗

- void **set_texture**(value: Texture3D)
- Texture3D **get_texture**()

The 3D texture representing the signed distance field.

---

float **thickness** = `1.0`

- void **set_thickness**(value: float)
- float **get_thickness**()

The collision shape's thickness. Unlike other particle colliders, **GPUParticlesCollisionSDF3D** is actually hollow on the inside. thickness can be increased to prevent particles from tunneling through the collision shape at high speeds, or when the **GPUParticlesCollisionSDF3D** is moved.

Method Descriptions

bool **get_bake_mask_value**(layer_number: int) const

Returns whether or not the specified layer of the bake_mask is enabled, given a `layer_number` between 1 and 32.

---

void **set_bake_mask_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the bake_mask, given a `layer_number` between 1 and 32.

GPUParticlesCollisionSphere3D

Inherits: GPUParticlesCollision3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A sphere-shaped 3D particle collision shape affecting GPUParticles3D nodes.

Description

A sphere-shaped 3D particle collision shape affecting GPUParticles3D nodes.

Particle collision shapes work in real-time and can be moved, rotated and scaled during gameplay. Unlike attractors, non-uniform scaling of collision shapes is *not* supported.

Note: ParticleProcessMaterial.collision_mode must be ParticleProcessMaterial.COLLISION_RIGID or ParticleProcessMaterial.COLLISION_HIDE_ON_CONTACT on the GPUParticles3D's process material for collision to work.

Note: Particle collision only affects GPUParticles3D, not CPUParticles3D.

Properties

float	radius	`1.0`

Property Descriptions

float **radius** = `1.0` 🔗

- void **set_radius**(value: float)
- float **get_radius**()

The collision sphere's radius in 3D units.

GraphEdit

Experimental: This class may be changed or removed in future versions.

Inherits: Control **<** CanvasItem **<** Node **<** Object

An editor for graph-like structures, using GraphNodes.

Description

GraphEdit provides tools for creation, manipulation, and display of various graphs. Its main purpose in the engine is to power the visual programming systems, such as visual shaders, but it is also available for use in user projects.

GraphEdit by itself is only an empty container, representing an infinite grid where GraphNodes can be placed. Each GraphNode represents a node in the graph, a single unit of data in the connected scheme. **GraphEdit**, in turn, helps to control various interactions with nodes and between nodes. When the user attempts to connect, disconnect, or delete a GraphNode, a signal is emitted in the **GraphEdit**, but no action is taken by default. It is the responsibility of the programmer utilizing this control to implement the necessary logic to determine how each request should be handled.

Performance: It is greatly advised to enable low-processor usage mode (see OS.low_processor_usage_mode) when using GraphEdits.

Note: Keep in mind that Node.get_children will also return the connection layer node named `_connection_layer` due to technical limitations. This behavior may change in future releases.

Properties

bool	clip_contents	`true` (overrides Control)
bool	connection_lines_antialiased	`true`
float	connection_lines_curvature	`0.5`
float	connection_lines_thickness	`4.0`
FocusMode	focus_mode	`2` (overrides Control)
GridPattern	grid_pattern	`0`
bool	minimap_enabled	`true`
float	minimap_opacity	`0.65`
Vector2	minimap_size	`Vector2(240, 160)`
PanningScheme	panning_scheme	`0`
bool	right_disconnects	`false`

Vector2	scroll_offset	`Vector2(0, 0)`
bool	show_arrange_button	`true`
bool	show_grid	`true`
bool	show_grid_buttons	`true`
bool	show_menu	`true`
bool	show_minimap_button	`true`
bool	show_zoom_buttons	`true`
bool	show_zoom_label	`false`
int	snapping_distance	`20`
bool	snapping_enabled	`true`
float	zoom	`1.0`
float	zoom_max	`2.0736`
float	zoom_min	`0.232568`

float	zoom_step	`1.2`

Methods

PackedVector2Array	_get_connection_line(from_position: Vector2, to_position: Vector2) virtual const
bool	_is_in_input_hotzone(in_node: Object, in_port: int, mouse_position: Vector2) virtual
bool	_is_in_output_hotzone(in_node: Object, in_port: int, mouse_position: Vector2) virtual
bool	_is_node_hover_valid(from_node: StringName, from_port: int, to_node: StringName, to_port: int) virtual
void	add_valid_connection_type(from_type: int, to_type: int)
void	add_valid_left_disconnect_type(type: int)
void	add_valid_right_disconnect_type(type: int)
void	arrange_nodes()

void	attach_graph_element_to_frame(element: StringName, frame: StringName)
void	clear_connections()
Error	connect_node(from_node: StringName, from_port: int, to_node: StringName, to_port: int)
void	detach_graph_element_from_frame(element: StringName)
void	disconnect_node(from_node: StringName, from_port: int, to_node: StringName, to_port: int)
void	force_connection_drag_end()
Array[StringName]	get_attached_nodes_of_frame(frame: StringName)
Dictionary	get_closest_connection_at_point(point: Vector2, max_distance: float = 4.0) const
PackedVector2Array	get_connection_line(from_node: Vector2, to_node: Vector2) const

Array[Dictionary]	get_connection_list() const
Array[Dictionary]	get_connections_intersecting_with_rect(rect: Rect2) const
GraphFrame	get_element_frame(element: StringName)
HBoxContainer	get_menu_hbox()
bool	is_node_connected(from_node: StringName, from_port: int, to_node: StringName, to_port: int)
bool	is_valid_connection_type(from_type: int, to_type: int) const
void	remove_valid_connection_type(from_type: int, to_type: int)
void	remove_valid_left_disconnect_type(type: int)
void	remove_valid_right_disconnect_type(type: int)
void	set_connection_activity(from_node: StringName, from_port: int, to_node: StringName, to_port: int, amount: float)

void	set_selected(node: Node)	

Theme Properties

Color	activity	`Color(1, 1, 1, 1)`
Color	connection_hover_tint_color	`Color(0, 0, 0, 0.3)`
Color	connection_rim_color	`Color(0.1, 0.1, 0.1, 0.6)`
Color	connection_valid_target_tint_color	`Color(1, 1, 1, 0.4)`
Color	grid_major	`Color(1, 1, 1, 0.2)`
Color	grid_minor	`Color(1, 1, 1, 0.05)`
Color	selection_fill	`Color(1, 1, 1, 0.3)`
Color	selection_stroke	`Color(1, 1, 1, 0.8)`

int	port_hotzone_inner_extent	`22`
int	port_hotzone_outer_extent	`26`
Texture2D	grid_toggle	
Texture2D	layout	
Texture2D	minimap_toggle	
Texture2D	snapping_toggle	
Texture2D	zoom_in	
Texture2D	zoom_out	
Texture2D	zoom_reset	
StyleBox	menu_panel	
StyleBox	panel	

Signals

begin_node_move()

Emitted at the beginning of a GraphElement's movement.

connection_drag_ended()

Emitted at the end of a connection drag.

connection_drag_started(from_node: StringName, from_port: int, is_output: bool)

Emitted at the beginning of a connection drag.

connection_from_empty(to_node: StringName, to_port: int, release_position: Vector2)

Emitted when user drags a connection from an input port into the empty space of the graph.

connection_request(from_node: StringName, from_port: int, to_node: StringName, to_port: int)

Emitted to the GraphEdit when the connection between the `from_port` of the `from_node` GraphNode and the `to_port` of the `to_node` GraphNode is attempted to be created.

connection_to_empty(from_node: StringName, from_port: int, release_position: Vector2)

Emitted when user drags a connection from an output port into the empty space of the graph.

copy_nodes_request()

Emitted when this **GraphEdit** captures a `ui_copy` action (`Ctrl + C` by default). In general, this signal indicates that the selected GraphElements should be copied.

---

delete_nodes_request(nodes: Array[StringName])

Emitted when this **GraphEdit** captures a `ui_graph_delete` action (`Delete` by default).

`nodes` is an array of node names that should be removed. These usually include all selected nodes.

---

disconnection_request(from_node: StringName, from_port: int, to_node: StringName, to_port: int)

Emitted to the GraphEdit when the connection between `from_port` of `from_node` GraphNode and `to_port` of `to_node` GraphNode is attempted to be removed.

---

duplicate_nodes_request()

Emitted when this **GraphEdit** captures a `ui_graph_duplicate` action (`Ctrl + D` by default). In general, this signal indicates that the selected GraphElements should be duplicated.

---

end_node_move()

Emitted at the end of a GraphElement's movement.

---

frame_rect_changed(frame: GraphFrame, new_rect: Vector2) 🔗

Emitted when the GraphFrame `frame` is resized to `new_rect`.

---

graph_elements_linked_to_frame_request(elements: Array, frame: StringName) 🔗

Emitted when one or more GraphElements are dropped onto the GraphFrame named `frame`, when they were not previously attached to any other one.

`elements` is an array of GraphElements to be attached.

---

node_deselected(node: Node) 🔗

Emitted when the given GraphElement node is deselected.

---

node_selected(node: Node) 🔗

Emitted when the given GraphElement node is selected.

---

paste_nodes_request() 🔗

Emitted when this **GraphEdit** captures a `ui_paste` action (`Ctrl + V` by default). In general, this signal indicates that previously copied GraphElements should be pasted.

---

popup_request(at_position: Vector2) 🔗

Emitted when a popup is requested. Happens on right-clicking in the GraphEdit. `at_position` is the position of the mouse pointer when the signal is sent.

---

scroll_offset_changed(offset: Vector2) 🔗

Emitted when the scroll offset is changed by the user. It will not be emitted when changed in code.

Enumerations

enum **PanningScheme**: 🔗

PanningScheme **SCROLL_ZOOMS** = `0`

`Mouse Wheel` will zoom, `Ctrl + Mouse Wheel` will move the view.

PanningScheme **SCROLL_PANS** = `1`

`Mouse Wheel` will move the view, `Ctrl + Mouse Wheel` will zoom.

---

enum **GridPattern**: 🔗

GridPattern **GRID_PATTERN_LINES** = `0`

Draw the grid using solid lines.

GridPattern **GRID_PATTERN_DOTS** = `1`

Draw the grid using dots.

Property Descriptions

bool **connection_lines_antialiased** = `true` 🔗

- void **set_connection_lines_antialiased**(value: bool)
- bool **is_connection_lines_antialiased**()

If `true`, the lines between nodes will use antialiasing.

---

float **connection_lines_curvature** = `0.5` 🔗

- void **set_connection_lines_curvature**(value: float)
- float **get_connection_lines_curvature**()

The curvature of the lines between the nodes. 0 results in straight lines.

---

float **connection_lines_thickness** = `4.0` 🔗

- void **set_connection_lines_thickness**(value: float)
- float **get_connection_lines_thickness**()

The thickness of the lines between the nodes.

GridPattern **grid_pattern** = `0` 🔗

- void **set_grid_pattern**(value: GridPattern)
- GridPattern **get_grid_pattern**()

The pattern used for drawing the grid.

---

bool **minimap_enabled** = `true` 🔗

- void **set_minimap_enabled**(value: bool)
- bool **is_minimap_enabled**()

If `true`, the minimap is visible.

---

float **minimap_opacity** = `0.65` 🔗

- void **set_minimap_opacity**(value: float)
- float **get_minimap_opacity**()

The opacity of the minimap rectangle.

Vector2 **minimap_size** = `Vector2(240, 160)` 🔗

- void **set_minimap_size**(value: Vector2)
- Vector2 **get_minimap_size**()

The size of the minimap rectangle. The map itself is based on the size of the grid area and is scaled to fit this rectangle.

PanningScheme **panning_scheme** = `0` 🔗

- void **set_panning_scheme**(value: PanningScheme)
- PanningScheme **get_panning_scheme**()

Defines the control scheme for panning with mouse wheel.

---

bool **right_disconnects** = `false` 🔗

- void **set_right_disconnects**(value: bool)
- bool **is_right_disconnects_enabled**()

If `true`, enables disconnection of existing connections in the GraphEdit by dragging the right end.

Vector2 **scroll_offset** = `Vector2(0, 0)` 🔗

- void **set_scroll_offset**(value: Vector2)
- Vector2 **get_scroll_offset**()

The scroll offset.

---

bool **show_arrange_button** = `true` 🔗

- void **set_show_arrange_button**(value: bool)
- bool **is_showing_arrange_button**()

If `true`, the button to automatically arrange graph nodes is visible.

---

bool **show_grid** = `true` 🔗

- void **set_show_grid**(value: bool)
- bool **is_showing_grid**()

If `true`, the grid is visible.

---

bool **show_grid_buttons** = `true` 🔗

- void **set_show_grid_buttons**(value: bool)
- bool **is_showing_grid_buttons**()

If `true`, buttons that allow to configure grid and snapping options are visible.

---

bool **show_menu** = `true` 🔗

- void **set_show_menu**(value: bool)
- bool **is_showing_menu**()

If `true`, the menu toolbar is visible.

---

bool **show_minimap_button** = `true`

- void **set_show_minimap_button**(value: bool)
- bool **is_showing_minimap_button**()

If `true`, the button to toggle the minimap is visible.

---

bool **show_zoom_buttons** = `true`

- void **set_show_zoom_buttons**(value: bool)
- bool **is_showing_zoom_buttons**()

If `true`, buttons that allow to change and reset the zoom level are visible.

---

bool **show_zoom_label** = `false`

- void **set_show_zoom_label**(value: bool)
- bool **is_showing_zoom_label**()

If `true`, the label with the current zoom level is visible. The zoom level is displayed in percents.

---

int **snapping_distance** = `20`

- void **set_snapping_distance**(value: int)
- int **get_snapping_distance**()

The snapping distance in pixels, also determines the grid line distance.

---

bool **snapping_enabled** = `true`

- void **set_snapping_enabled**(value: bool)
- bool **is_snapping_enabled**()

If `true`, enables snapping.

float **zoom** = `1.0`

- void **set_zoom**(value: float)
- float **get_zoom**()

The current zoom value.

float **zoom_max** = `2.0736`

- void **set_zoom_max**(value: float)
- float **get_zoom_max**()

The upper zoom limit.

float **zoom_min** = `0.232568`

- void **set_zoom_min**(value: float)
- float **get_zoom_min**()

The lower zoom limit.

float **zoom_step** = `1.2`

- void **set_zoom_step**(value: float)
- float **get_zoom_step**()

The step of each zoom level.

Method Descriptions

PackedVector2Array **_get_connection_line**(from_position: Vector2, to_position: Vector2) virtual const 🔗

Virtual method which can be overridden to customize how connections are drawn.

---

bool **_is_in_input_hotzone**(in_node: Object, in_port: int, mouse_position: Vector2) virtual 🔗

Returns whether the `mouse_position` is in the input hot zone.

By default, a hot zone is a Rect2 positioned such that its center is at `in_node`.GraphNode.get_input_port_position(`in_port`) (For output's case, call GraphNode.get_output_port_position instead). The hot zone's width is twice the Theme Property `port_grab_distance_horizontal`, and its height is twice the `port_grab_distance_vertical`.

Below is a sample code to help get started:

```
func _is_in_input_hotzone(in_node, in_port, mouse_position):
    var port_size: Vector2 =
Vector2(get_theme_constant("port_grab_distance_horizontal"),
get_theme_constant("port_grab_distance_vertical"))
    var port_pos: Vector2 = in_node.get_position() +
in_node.get_input_port_position(in_port) - port_size / 2
    var rect = Rect2(port_pos, port_size)

    return rect.has_point(mouse_position)
```

---

bool **_is_in_output_hotzone**(in_node: Object, in_port: int, mouse_position: Vector2) virtual 🔗

Returns whether the `mouse_position` is in the output hot zone. For more information on hot zones, see _is_in_input_hotzone.

Below is a sample code to help get started:

```
func _is_in_output_hotzone(in_node, in_port, mouse_position):
    var port_size: Vector2 =
Vector2(get_theme_constant("port_grab_distance_horizontal"),
get_theme_constant("port_grab_distance_vertical"))
    var port_pos: Vector2 = in_node.get_position() +
in_node.get_output_port_position(in_port) - port_size / 2
    var rect = Rect2(port_pos, port_size)

    return rect.has_point(mouse_position)
```

---

bool **_is_node_hover_valid**(from_node: StringName, from_port: int, to_node: StringName, to_port: int) virtual 🔗

This virtual method can be used to insert additional error detection while the user is dragging a connection over a valid port.

Return `true` if the connection is indeed valid or return `false` if the connection is impossible. If the connection is impossible, no snapping to the port and thus no connection request to that port will happen.

In this example a connection to same node is suppressed:

GDScript

```
func _is_node_hover_valid(from, from_port, to, to_port):
    return from != to
```

C#

```
public override bool _IsNodeHoverValid(StringName fromNode, int
fromPort, StringName toNode, int toPort)
{
    return fromNode != toNode;
}
```

---

void **add_valid_connection_type**(from_type: int, to_type: int)

Allows the connection between two different port types. The port type is defined individually for the left and the right port of each slot with the GraphNode.set_slot method.

See also is_valid_connection_type and remove_valid_connection_type.

---

void **add_valid_left_disconnect_type**(type: int)

Allows to disconnect nodes when dragging from the left port of the GraphNode's slot if it has the specified type. See also remove_valid_left_disconnect_type.

---

void **add_valid_right_disconnect_type**(type: int)

Allows to disconnect nodes when dragging from the right port of the GraphNode's slot if it has the specified type. See also remove_valid_right_disconnect_type.

---

void **arrange_nodes**()

Rearranges selected nodes in a layout with minimum crossings between connections and uniform horizontal and vertical gap between nodes.

---

void **attach_graph_element_to_frame**(element: StringName, frame: StringName) 🔗

Attaches the `element` GraphElement to the `frame` GraphFrame.

---

void **clear_connections**() 🔗

Removes all connections between nodes.

Error **connect_node**(from_node: StringName, from_port: int, to_node: StringName, to_port: int) 🔗

Create a connection between the `from_port` of the `from_node` GraphNode and the `to_port` of the `to_node` GraphNode. If the connection already exists, no connection is created.

---

void **detach_graph_element_from_frame**(element: StringName) 🔗

Detaches the `element` GraphElement from the GraphFrame it is currently attached to.

---

void **disconnect_node**(from_node: StringName, from_port: int, to_node: StringName, to_port: int) 🔗

Removes the connection between the `from_port` of the `from_node` GraphNode and the `to_port` of the `to_node` GraphNode. If the connection does not exist, no connection is removed.

---

void **force_connection_drag_end**() 🔗

Ends the creation of the current connection. In other words, if you are dragging a connection you can use this method to abort the process and remove the line that followed your cursor.

This is best used together with connection_drag_started and connection_drag_ended to add custom behavior like node addition through shortcuts.

Note: This method suppresses any other connection request signals apart from connection_drag_ended.

Array[StringName] **get_attached_nodes_of_frame**(frame: StringName) 🔗

Returns an array of node names that are attached to the GraphFrame with the given name.

Dictionary **get_closest_connection_at_point**(point: Vector2, max_distance: float = 4.0) const 🔗

Returns the closest connection to the given point in screen space. If no connection is found within `max_distance` pixels, an empty Dictionary is returned.

A connection consists in a structure of the form `{ from_port: 0, from_node: "GraphNode name 0", to_port: 1, to_node: "GraphNode name 1" }`.

For example, getting a connection at a given mouse position can be achieved like this:

GDScript

```
var connection =
get_closest_connection_at_point(mouse_event.get_position())
```

PackedVector2Array **get_connection_line**(from_node: Vector2, to_node: Vector2) const 🔗

Returns the points which would make up a connection between `from_node` and `to_node`.

Array[Dictionary] **get_connection_list**() const 🔗

Returns an Array containing the list of connections. A connection consists in a structure of the form `{ from_port: 0, from_node: "GraphNode name 0", to_port: 1, to_node: "GraphNode name 1" }`.

Array[Dictionary] **get_connections_intersecting_with_rect**(rect: Rect2) const 🔗

Returns an Array containing the list of connections that intersect with the given Rect2. A connection consists in a structure of the form `{ from_port: 0, from_node: "GraphNode name 0", to_port: 1, to_node: "GraphNode name 1" }`.

GraphFrame **get_element_frame**(element: StringName) 🔗

Returns the GraphFrame that contains the GraphElement with the given name.

HBoxContainer **get_menu_hbox**() 🔗

Gets the HBoxContainer that contains the zooming and grid snap controls in the top left of the graph. You can use this method to reposition the toolbar or to add your own custom controls to it.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their CanvasItem.visible property.

---

bool **is_node_connected**(from_node: StringName, from_port: int, to_node: StringName, to_port: int)

Returns `true` if the `from_port` of the `from_node` GraphNode is connected to the `to_port` of the `to_node` GraphNode.

---

bool **is_valid_connection_type**(from_type: int, to_type: int) const

Returns whether it's possible to make a connection between two different port types. The port type is defined individually for the left and the right port of each slot with the GraphNode.set_slot method.

See also add_valid_connection_type and remove_valid_connection_type.

---

void **remove_valid_connection_type**(from_type: int, to_type: int)

Disallows the connection between two different port types previously allowed by add_valid_connection_type. The port type is defined individually for the left and the right port of each slot with the GraphNode.set_slot method.

See also is_valid_connection_type.

---

void **remove_valid_left_disconnect_type**(type: int)

Disallows to disconnect nodes when dragging from the left port of the GraphNode's slot if it has the specified type. Use this to disable disconnection previously allowed with add_valid_left_disconnect_type.

---

void **remove_valid_right_disconnect_type**(type: int) 🔗

Disallows to disconnect nodes when dragging from the right port of the GraphNode's slot if it has the specified type. Use this to disable disconnection previously allowed with add_valid_right_disconnect_type.

---

void **set_connection_activity**(from_node: StringName, from_port: int, to_node: StringName, to_port: int, amount: float) 🔗

Sets the coloration of the connection between `from_node`'s `from_port` and `to_node`'s `to_port` with the color provided in the activity theme property. The color is linearly interpolated between the connection color and the activity color using `amount` as weight.

---

void **set_selected**(node: Node) 🔗

Sets the specified `node` as the one selected.

Theme Property Descriptions

Color **activity** = `Color(1, 1, 1, 1)` 🔗

Color the connection line is interpolated to based on the activity value of a connection (see set_connection_activity).

Color **connection_hover_tint_color** = `Color(0, 0, 0, 0.3)` 🔗

Color which is blended with the connection line when the mouse is hovering over it.

Color **connection_rim_color** = `Color(0.1, 0.1, 0.1, 0.6)` 🔗

Color of the rim around each connection line used for making intersecting lines more distinguishable.

Color **connection_valid_target_tint_color** = `Color(1, 1, 1, 0.4)` 🔗

Color which is blended with the connection line when the currently dragged connection is hovering over a valid target port.

Color **grid_major** = `Color(1, 1, 1, 0.2)` 🔗

Color of major grid lines/dots.

Color **grid_minor** = `Color(1, 1, 1, 0.05)` 🔗

Color of minor grid lines/dots.

Color **selection_fill** = `Color(1, 1, 1, 0.3)` 🔗

The fill color of the selection rectangle.

Color **selection_stroke** = `Color(1, 1, 1, 0.8)` 🔗

The outline color of the selection rectangle.

int **port_hotzone_inner_extent** = `22`

The horizontal range within which a port can be grabbed (inner side).

int **port_hotzone_outer_extent** = `26`

The horizontal range within which a port can be grabbed (outer side).

Texture2D **grid_toggle**

The icon for the grid toggle button.

Texture2D **layout**

The icon for the layout button for auto-arranging the graph.

Texture2D **minimap_toggle**

The icon for the minimap toggle button.

Texture2D **snapping_toggle**

The icon for the snapping toggle button.

Texture2D **zoom_in**

The icon for the zoom in button.

Texture2D **zoom_out** 🔗

The icon for the zoom out button.

Texture2D **zoom_reset** 🔗

The icon for the zoom reset button.

StyleBox **menu_panel** 🔗

There is currently no description for this theme property. Please help us by contributing one!

StyleBox **panel** 🔗

The background drawn under the grid.

GraphElement

Experimental: This class may be changed or removed in future versions.

Inherits: Container **<** Control **<** CanvasItem **<** Node **<** Object

Inherited By: GraphFrame, GraphNode

A container that represents a basic element that can be placed inside a GraphEdit control.

Description

GraphElement allows to create custom elements for a GraphEdit graph. By default such elements can be selected, resized, and repositioned, but they cannot be connected. For a graph element that allows for connections see GraphNode.

Properties

bool	draggable	`true`
Vector2	position_offset	`Vector2(0, 0)`
bool	resizable	`false`
bool	selectable	`true`

bool	selected	`false`

Theme Properties

Texture2D	resizer

Signals

delete_request() 🔗

Emitted when removing the GraphElement is requested.

---

dragged(from: Vector2, to: Vector2) 🔗

Emitted when the GraphElement is dragged.

---

node_deselected() 🔗

Emitted when the GraphElement is deselected.

---

node_selected() 🔗

Emitted when the GraphElement is selected.

---

position_offset_changed() 🔗

Emitted when the GraphElement is moved.

---

raise_request() 🔗

Emitted when displaying the GraphElement over other ones is requested. Happens on focusing (clicking into) the GraphElement.

---

resize_end(new_size: Vector2) 🔗

Emitted when releasing the mouse button after dragging the resizer handle (see resizable).

---

resize_request(new_size: Vector2) 🔗

Emitted when resizing the GraphElement is requested. Happens on dragging the resizer handle (see resizable).

Property Descriptions

bool **draggable** = `true` 🔗

- void **set_draggable**(value: bool)
- bool **is_draggable**()

If `true`, the user can drag the GraphElement.

Vector2 **position_offset** = `Vector2(0, 0)` 🔗

- void **set_position_offset**(value: Vector2)
- Vector2 **get_position_offset**()

The offset of the GraphElement, relative to the scroll offset of the GraphEdit.

---

bool **resizable** = `false`

- void **set_resizable**(value: bool)
- bool **is_resizable**()

If `true`, the user can resize the GraphElement.

Note: Dragging the handle will only emit the resize_request and resize_end signals, the GraphElement needs to be resized manually.

---

bool **selectable** = `true`

- void **set_selectable**(value: bool)
- bool **is_selectable**()

If `true`, the user can select the GraphElement.

---

bool **selected** = `false`

- void **set_selected**(value: bool)
- bool **is_selected**()

If `true`, the GraphElement is selected.

Theme Property Descriptions

Texture2D **resizer**

The icon used for the resizer, visible when resizable is enabled.

GraphFrame

Experimental: This class may be changed or removed in future versions.

Inherits: GraphElement **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

GraphFrame is a special GraphElement that can be used to organize other GraphElements inside a GraphEdit.

Description

GraphFrame is a special GraphElement to which other GraphElements can be attached. It can be configured to automatically resize to enclose all attached GraphElements. If the frame is moved, all the attached GraphElements inside it will be moved as well.

A GraphFrame is always kept behind the connection layer and other GraphElements inside a GraphEdit.

Properties

bool	autoshrink_enabled	`true`
int	autoshrink_margin	`40`
int	drag_margin	`16`

MouseFilter	mouse_filter	`0` (overrides Control)
Color	tint_color	`Color(0.3, 0.3, 0.3, 0.75)`
bool	tint_color_enabled	`false`
String	title	`""`

Methods

HBoxContainer	get_titlebar_hbox()

Theme Properties

Color	resizer_color	`Color(0.875, 0.875, 0.875, 1)`
StyleBox	panel	
StyleBox	panel_selected	
StyleBox	titlebar	
StyleBox	titlebar_selected	

Signals

autoshrink_changed() 🔗

Emitted when autoshrink_enabled or autoshrink_margin changes.

Property Descriptions

bool **autoshrink_enabled** = `true` 🔗

- void **set_autoshrink_enabled**(value: bool)
- bool **is_autoshrink_enabled**()

If `true`, the frame's rect will be adjusted automatically to enclose all attached GraphElements.

---

int **autoshrink_margin** = `40` 🔗

- void **set_autoshrink_margin**(value: int)
- int **get_autoshrink_margin**()

The margin around the attached nodes that is used to calculate the size of the frame when autoshrink_enabled is `true`.

---

int **drag_margin** = `16` 🔗

- void **set_drag_margin**(value: int)
- int **get_drag_margin**()

The margin inside the frame that can be used to drag the frame.

Color **tint_color** = `Color(0.3, 0.3, 0.3, 0.75)`

- void **set_tint_color**(value: Color)
- Color **get_tint_color**()

The color of the frame when tint_color_enabled is `true`.

---

bool **tint_color_enabled** = `false`

- void **set_tint_color_enabled**(value: bool)
- bool **is_tint_color_enabled**()

If `true`, the tint color will be used to tint the frame.

String **title** = `""`

- void **set_title**(value: String)
- String **get_title**()

Title of the frame.

Method Descriptions

HBoxContainer **get_titlebar_hbox**()

Returns the HBoxContainer used for the title bar, only containing a Label for displaying the title by default.

This can be used to add custom controls to the title bar such as option or close buttons.

Theme Property Descriptions

Color **resizer_color** = `Color(0.875, 0.875, 0.875, 1)`

The color modulation applied to the resizer icon.

StyleBox **panel**

The default StyleBox used for the background of the **GraphFrame**.

StyleBox **panel_selected**

The StyleBox used for the background of the **GraphFrame** when it is selected.

StyleBox **titlebar**

The StyleBox used for the title bar of the **GraphFrame**.

StyleBox **titlebar_selected**

The StyleBox used for the title bar of the **GraphFrame** when it is selected.

GraphNode

Experimental: This class may be changed or removed in future versions.

Inherits: GraphElement **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

A container with connection ports, representing a node in a GraphEdit.

Description

GraphNode allows to create nodes for a GraphEdit graph with customizable content based on its child controls. **GraphNode** is derived from Container and it is responsible for placing its children on screen. This works similar to VBoxContainer. Children, in turn, provide **GraphNode** with so-called slots, each of which can have a connection port on either side.

Each **GraphNode** slot is defined by its index and can provide the node with up to two ports: one on the left, and one on the right. By convention the left port is also referred to as the **input port** and the right port is referred to as the **output port**. Each port can be enabled and configured individually, using different type and color. The type is an arbitrary value that you can define using your own considerations. The parent GraphEdit will receive this information on each connect and disconnect request.

Slots can be configured in the Inspector dock once you add at least one child Control. The properties are grouped by each slot's index in the "Slot" section.

Note: While GraphNode is set up using slots and slot indices, connections are made between the ports which are enabled. Because of that GraphEdit uses the port's index and not the slot's index. You can use get_input_port_slot and get_output_port_slot to get the slot index from the port index.

Properties

bool	ignore_invalid_connection_type	`false`
MouseFilter	mouse_filter	`0` (overrides Control)
String	title	`""`

Methods

void	_draw_port(slot_index: int, position: Vector2i, left: bool, color: Color) virtual
void	clear_all_slots()
void	clear_slot(slot_index: int)
Color	get_input_port_color(port_idx: int)
int	get_input_port_count()

Vector2	get_input_port_position(port_idx: int)
int	get_input_port_slot(port_idx: int)
int	get_input_port_type(port_idx: int)
Color	get_output_port_color(port_idx: int)
int	get_output_port_count()
Vector2	get_output_port_position(port_idx: int)
int	get_output_port_slot(port_idx: int)
int	get_output_port_type(port_idx: int)
Color	get_slot_color_left(slot_index: int) const
Color	get_slot_color_right(slot_index: int) const
Texture2D	get_slot_custom_icon_left(slot_index: int) const
Texture2D	get_slot_custom_icon_right(slot_index: int) const
int	get_slot_type_left(slot_index: int) const

int	get_slot_type_right(slot_index: int) const
HBoxContainer	get_titlebar_hbox()
bool	is_slot_draw_stylebox(slot_index: int) const
bool	is_slot_enabled_left(slot_index: int) const
bool	is_slot_enabled_right(slot_index: int) const
void	set_slot(slot_index: int, enable_left_port: bool, type_left: int, color_left: Color, enable_right_port: bool, type_right: int, color_right: Color, custom_icon_left: Texture2D = null, custom_icon_right: Texture2D = null, draw_stylebox: bool = true)
void	set_slot_color_left(slot_index: int, color: Color)
void	set_slot_color_right(slot_index: int, color: Color)
void	set_slot_custom_icon_left(slot_index: int, custom_icon: Texture2D)
void	set_slot_custom_icon_right(slot_index: int, custom_icon: Texture2D)

void	set_slot_draw_stylebox(slot_index: int, enable: bool)
void	set_slot_enabled_left(slot_index: int, enable: bool)
void	set_slot_enabled_right(slot_index: int, enable: bool)
void	set_slot_type_left(slot_index: int, type: int)
void	set_slot_type_right(slot_index: int, type: int)

Theme Properties

Color	resizer_color	`Color(0.875, 0.875, 0.875, 1)`
int	port_h_offset	`0`
int	separation	`2`
Texture2D	port	
StyleBox	panel	
StyleBox	panel_selected	

StyleBox	slot
StyleBox	titlebar
StyleBox	titlebar_selected

Signals

slot_updated(slot_index: int)

Emitted when any GraphNode's slot is updated.

Property Descriptions

bool **ignore_invalid_connection_type** = `false`

- void **set_ignore_invalid_connection_type**(value: bool)
- bool **is_ignoring_valid_connection_type**()

If `true`, you can connect ports with different types, even if the connection was not explicitly allowed in the parent GraphEdit.

String **title** = `""`

- void **set_title**(value: String)
- String **get_title**()

The text displayed in the GraphNode's title bar.

Method Descriptions

void **_draw_port**(slot_index: int, position: Vector2i, left: bool, color: Color) virtual 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **clear_all_slots**() 🔗

Disables all slots of the GraphNode. This will remove all input/output ports from the GraphNode.

---

void **clear_slot**(slot_index: int) 🔗

Disables the slot with the given `slot_index`. This will remove the corresponding input and output port from the GraphNode.

Color **get_input_port_color**(port_idx: int) 🔗

Returns the Color of the input port with the given `port_idx`.

---

int **get_input_port_count**() 🔗

Returns the number of slots with an enabled input port.

Vector2 **get_input_port_position**(port_idx: int) 🔗

Returns the position of the input port with the given `port_idx`.

---

int **get_input_port_slot**(port_idx: int)

Returns the corresponding slot index of the input port with the given `port_idx`.

---

int **get_input_port_type**(port_idx: int)

Returns the type of the input port with the given `port_idx`.

Color **get_output_port_color**(port_idx: int)

Returns the Color of the output port with the given `port_idx`.

---

int **get_output_port_count**()

Returns the number of slots with an enabled output port.

Vector2 **get_output_port_position**(port_idx: int)

Returns the position of the output port with the given `port_idx`.

---

int **get_output_port_slot**(port_idx: int)

Returns the corresponding slot index of the output port with the given `port_idx`.

---

int **get_output_port_type**(port_idx: int)

Returns the type of the output port with the given `port_idx`.

Color **get_slot_color_left**(slot_index: int) const

Returns the left (input) Color of the slot with the given `slot_index`.

Color **get_slot_color_right**(slot_index: int) const

Returns the right (output) Color of the slot with the given `slot_index`.

Texture2D **get_slot_custom_icon_left**(slot_index: int) const

Returns the left (input) custom Texture2D of the slot with the given `slot_index`.

Texture2D **get_slot_custom_icon_right**(slot_index: int) const

Returns the right (output) custom Texture2D of the slot with the given `slot_index`.

---

int **get_slot_type_left**(slot_index: int) const

Returns the left (input) type of the slot with the given `slot_index`.

---

int **get_slot_type_right**(slot_index: int) const

Returns the right (output) type of the slot with the given `slot_index`.

HBoxContainer **get_titlebar_hbox**()

Returns the HBoxContainer used for the title bar, only containing a Label for displaying the title by default. This can be used to add custom controls to the title bar such as option or close buttons.

---

bool **is_slot_draw_stylebox**(slot_index: int) const 🔗

Returns true if the background StyleBox of the slot with the given `slot_index` is drawn.

---

bool **is_slot_enabled_left**(slot_index: int) const 🔗

Returns `true` if left (input) side of the slot with the given `slot_index` is enabled.

---

bool **is_slot_enabled_right**(slot_index: int) const 🔗

Returns `true` if right (output) side of the slot with the given `slot_index` is enabled.

---

void **set_slot**(slot_index: int, enable_left_port: bool, type_left: int, color_left: Color, enable_right_port: bool, type_right: int, color_right: Color, custom_icon_left: Texture2D = null, custom_icon_right: Texture2D = null, draw_stylebox: bool = true) 🔗

Sets properties of the slot with the given `slot_index`.

If `enable_left_port`/`enable_right_port` is `true`, a port will appear and the slot will be able to be connected from this side.

With `type_left`/`type_right` an arbitrary type can be assigned to each port. Two ports can be connected if they share the same type, or if the connection between their types is allowed in the parent

GraphEdit (see GraphEdit.add_valid_connection_type). Keep in mind that the GraphEdit has the final say in accepting the connection. Type compatibility simply allows the GraphEdit.connection_request signal to be emitted.

Ports can be further customized using `color_left`/`color_right` and `custom_icon_left`/`custom_icon_right`. The color parameter adds a tint to the icon. The custom icon can be used to override the default port dot.

Additionally, `draw_stylebox` can be used to enable or disable drawing of the background stylebox for each slot. See slot.

Individual properties can also be set using one of the `set_slot_*` methods.

Note: This method only sets properties of the slot. To create the slot itself, add a Control-derived child to the GraphNode.

---

void **set_slot_color_left**(slot_index: int, color: Color) 🔗

Sets the Color of the left (input) side of the slot with the given `slot_index` to `color`.

---

void **set_slot_color_right**(slot_index: int, color: Color) 🔗

Sets the Color of the right (output) side of the slot with the given `slot_index` to `color`.

---

void **set_slot_custom_icon_left**(slot_index: int, custom_icon: Texture2D) 🔗

Sets the custom Texture2D of the left (input) side of the slot with the given `slot_index` to `custom_icon`.

---

void **set_slot_custom_icon_right**(slot_index: int, custom_icon: Texture2D)

Sets the custom Texture2D of the right (output) side of the slot with the given `slot_index` to `custom_icon`.

---

void **set_slot_draw_stylebox**(slot_index: int, enable: bool)

Toggles the background StyleBox of the slot with the given `slot_index`.

---

void **set_slot_enabled_left**(slot_index: int, enable: bool)

Toggles the left (input) side of the slot with the given `slot_index`. If `enable` is `true`, a port will appear on the left side and the slot will be able to be connected from this side.

---

void **set_slot_enabled_right**(slot_index: int, enable: bool)

Toggles the right (output) side of the slot with the given `slot_index`. If `enable` is `true`, a port will appear on the right side and the slot will be able to be connected from this side.

---

void **set_slot_type_left**(slot_index: int, type: int)

Sets the left (input) type of the slot with the given `slot_index` to `type`. If the value is negative, all connections will be disallowed to be created via user inputs.

---

void **set_slot_type_right**(slot_index: int, type: int)

Sets the right (output) type of the slot with the given `slot_index` to `type`. If the value is negative, all connections will be disallowed to be created via user inputs.

Theme Property Descriptions

Color **resizer_color** = `Color(0.875, 0.875, 0.875, 1)`

The color modulation applied to the resizer icon.

---

int **port_h_offset** = `0`

Horizontal offset for the ports.

---

int **separation** = `2`

The vertical distance between ports.

Texture2D **port**

The icon used for representing ports.

StyleBox **panel**

The default background for the slot area of the **GraphNode**.

StyleBox **panel_selected**

The StyleBox used for the slot area when selected.

StyleBox **slot**

The StyleBox used for each slot of the **GraphNode**.

StyleBox **titlebar**

The StyleBox used for the title bar of the **GraphNode**.

StyleBox **titlebar_selected**

The StyleBox used for the title bar of the **GraphNode** when it is selected.

GridContainer

Inherits: Container **<** Control **<** CanvasItem **<** Node **<** Object

A container that arranges its child controls in a grid layout.

Description

GridContainer arranges its child controls in a grid layout. The number of columns is specified by the columns property, whereas the number of rows depends on how many are needed for the child controls. The number of rows and columns is preserved for every size of the container.

Note: GridContainer only works with child nodes inheriting from Control. It won't rearrange child nodes inheriting from Node2D.

Tutorials

- Using Containers
- Operating System Testing Demo [https://godotengine.org/asset-library/asset/2789]

Properties

int	columns	`1`

Theme Properties

int	h_separation	4
int	v_separation	4

Property Descriptions

int **columns** = 1

- void **set_columns**(value: int)
- int **get_columns**()

The number of columns in the **GridContainer**. If modified, **GridContainer** reorders its Control-derived children to accommodate the new layout.

Theme Property Descriptions

int **h_separation** = 4

The horizontal separation of child nodes.

---

int **v_separation** = 4

The vertical separation of child nodes.

GridMap

Inherits: Node3D **<** Node **<** Object

Node for 3D tile-based maps.

Description

GridMap lets you place meshes on a grid interactively. It works both from the editor and from scripts, which can help you create in-game level editors.

GridMaps use a MeshLibrary which contains a list of tiles. Each tile is a mesh with materials plus optional collision and navigation shapes.

A GridMap contains a collection of cells. Each grid cell refers to a tile in the MeshLibrary. All cells in the map have the same dimensions.

Internally, a GridMap is split into a sparse collection of octants for efficient rendering and physics processing. Every octant has the same dimensions and can contain several cells.

Note: GridMap doesn't extend VisualInstance3D and therefore can't be hidden or cull masked based on VisualInstance3D.layers. If you make a light not affect the first layer, the whole GridMap won't be lit by the light in question.

Tutorials

- Using gridmaps
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]

- 3D Kinematic Character Demo [https://godotengine.org/asset-library/asset/2739]

Properties

bool	bake_navigation	`false`
bool	cell_center_x	`true`
bool	cell_center_y	`true`
bool	cell_center_z	`true`
int	cell_octant_size	`8`
float	cell_scale	`1.0`
Vector3	cell_size	`Vector3(2, 2, 2)`
int	collision_layer	`1`
int	collision_mask	`1`
float	collision_priority	`1.0`

MeshLibrary	mesh_library
PhysicsMaterial	physics_material

Methods

void	clear()
void	clear_baked_meshes()
RID	get_bake_mesh_instance(idx: int)
Array	get_bake_meshes()
Basis	get_basis_with_orthogonal_index(index: int) const
int	get_cell_item(position: Vector3i) const
Basis	get_cell_item_basis(position: Vector3i) const
int	get_cell_item_orientation(position: Vector3i) const
bool	get_collision_layer_value(layer_number: int) const

bool	get_collision_mask_value(layer_number: int) const
Array	get_meshes() const
RID	get_navigation_map() const
int	get_orthogonal_index_from_basis(basis: Basis) const
Array[Vector3i]	get_used_cells() const
Array[Vector3i]	get_used_cells_by_item(item: int) const
Vector3i	local_to_map(local_position: Vector3) const
void	make_baked_meshes(gen_lightmap_uv: bool = false, lightmap_uv_texel_size: float = 0.1)
Vector3	map_to_local(map_position: Vector3i) const
void	resource_changed(resource: Resource)
void	set_cell_item(position: Vector3i, item: int, orientation: int = 0)

void	set_collision_layer_value(layer_number: int, value: bool)
void	set_collision_mask_value(layer_number: int, value: bool)
void	set_navigation_map(navigation_map: RID)

Signals

cell_size_changed(cell_size: Vector3) 🔗

Emitted when cell_size changes.

changed() 🔗

Emitted when the MeshLibrary of this GridMap changes.

Constants

INVALID_CELL_ITEM = `-1` 🔗

Invalid cell item that can be used in set_cell_item to clear cells (or represent an empty cell in get_cell_item).

Property Descriptions

bool **bake_navigation** = `false`

- void **set_bake_navigation**(value: bool)
- bool **is_baking_navigation**()

If `true`, this GridMap creates a navigation region for each cell that uses a mesh_library item with a navigation mesh. The created navigation region will use the navigation layers bitmask assigned to the MeshLibrary's item.

---

bool **cell_center_x** = `true`

- void **set_center_x**(value: bool)
- bool **get_center_x**()

If `true`, grid items are centered on the X axis.

---

bool **cell_center_y** = `true`

- void **set_center_y**(value: bool)
- bool **get_center_y**()

If `true`, grid items are centered on the Y axis.

---

bool **cell_center_z** = `true`

- void **set_center_z**(value: bool)
- bool **get_center_z**()

If `true`, grid items are centered on the Z axis.

---

int **cell_octant_size** = `8`

- void **set_octant_size**(value: int)
- int **get_octant_size**()

The size of each octant measured in number of cells. This applies to all three axis.

---

float **cell_scale** = `1.0` 🔗

- void **set_cell_scale**(value: float)
- float **get_cell_scale**()

The scale of the cell items.

This does not affect the size of the grid cells themselves, only the items in them. This can be used to make cell items overlap their neighbors.

Vector3 **cell_size** = `Vector3(2, 2, 2)` 🔗

- void **set_cell_size**(value: Vector3)
- Vector3 **get_cell_size**()

The dimensions of the grid's cells.

This does not affect the size of the meshes. See cell_scale.

---

int **collision_layer** = `1` 🔗

- void **set_collision_layer**(value: int)
- int **get_collision_layer**()

The physics layers this GridMap is in.

GridMaps act as static bodies, meaning they aren't affected by gravity or other forces. They only affect other physics bodies that collide with them.

---

int **collision_mask** = `1` 🔗

- void **set_collision_mask**(value: int)
- int **get_collision_mask**()

The physics layers this GridMap detects collisions in. See Collision layers and masks in the documentation for more information.

---

float **collision_priority** = `1.0` 🔗

- void **set_collision_priority**(value: float)
- float **get_collision_priority**()

The priority used to solve colliding when occurring penetration. The higher the priority is, the lower the penetration into the object will be. This can for example be used to prevent the player from breaking through the boundaries of a level.

MeshLibrary **mesh_library** 🔗

- void **set_mesh_library**(value: MeshLibrary)
- MeshLibrary **get_mesh_library**()

The assigned MeshLibrary.

PhysicsMaterial **physics_material** 🔗

- void **set_physics_material**(value: PhysicsMaterial)
- PhysicsMaterial **get_physics_material**()

Overrides the default friction and bounce physics properties for the whole **GridMap**.

Method Descriptions

void **clear**()

Clear all cells.

---

void **clear_baked_meshes**()

Clears all baked meshes. See make_baked_meshes.

RID **get_bake_mesh_instance**(idx: int)

Returns RID of a baked mesh with the given `idx`.

Array **get_bake_meshes**()

Returns an array of ArrayMeshes and Transform3D references of all bake meshes that exist within the current GridMap.

Basis **get_basis_with_orthogonal_index**(index: int) const

Returns one of 24 possible rotations that lie along the vectors (x,y,z) with each component being either -1, 0, or 1. For further details, refer to the Godot source code.

---

int **get_cell_item**(position: Vector3i) const

The MeshLibrary item index located at the given grid coordinates. If the cell is empty, INVALID_CELL_ITEM will be returned.

Basis **get_cell_item_basis**(position: Vector3i) const 🔗

Returns the basis that gives the specified cell its orientation.

---

int **get_cell_item_orientation**(position: Vector3i) const 🔗

The orientation of the cell at the given grid coordinates. `-1` is returned if the cell is empty.

---

bool **get_collision_layer_value**(layer_number: int) const 🔗

Returns whether or not the specified layer of the collision_layer is enabled, given a `layer_number` between 1 and 32.

---

bool **get_collision_mask_value**(layer_number: int) const 🔗

Returns whether or not the specified layer of the collision_mask is enabled, given a `layer_number` between 1 and 32.

Array **get_meshes**() const 🔗

Returns an array of Transform3D and Mesh references corresponding to the non-empty cells in the grid. The transforms are specified in local space.

RID **get_navigation_map**() const 🔗

Returns the RID of the navigation map this GridMap node uses for its cell baked navigation meshes.

This function returns always the map set on the GridMap node and not the map on the NavigationServer. If the map is changed directly with the NavigationServer API the GridMap node will not be aware of the map change.

---

int **get_orthogonal_index_from_basis**(basis: Basis) const 🔗

This function considers a discretization of rotations into 24 points on unit sphere, lying along the vectors (x,y,z) with each component being either -1, 0, or 1, and returns the index (in the range from 0 to 23) of the point best representing the orientation of the object. For further details, refer to the Godot source code.

Array[Vector3i] **get_used_cells**() const 🔗

Returns an array of Vector3 with the non-empty cell coordinates in the grid map.

Array[Vector3i] **get_used_cells_by_item**(item: int) const 🔗

Returns an array of all cells with the given item index specified in `item`.

Vector3i **local_to_map**(local_position: Vector3) const 🔗

Returns the map coordinates of the cell containing the given `local_position`. If `local_position` is in global coordinates, consider using Node3D.to_local before passing it to this method. See also map_to_local.

---

void **make_baked_meshes**(gen_lightmap_uv: bool = false, lightmap_uv_texel_size: float = 0.1)

Bakes lightmap data for all meshes in the assigned MeshLibrary.

Vector3 **map_to_local**(map_position: Vector3i) const

Returns the position of a grid cell in the GridMap's local coordinate space. To convert the returned value into global coordinates, use Node3D.to_global. See also local_to_map.

---

void **resource_changed**(resource: Resource)

Deprecated: Use Resource.changed instead.

This method does nothing.

---

void **set_cell_item**(position: Vector3i, item: int, orientation: int = 0)

Sets the mesh index for the cell referenced by its grid coordinates.

A negative item index such as INVALID_CELL_ITEM will clear the cell.

Optionally, the item's orientation can be passed. For valid orientation values, see get_orthogonal_index_from_basis.

---

void **set_collision_layer_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the collision_layer, given a `layer_number` between 1 and 32.

---

void **set_collision_mask_value**(layer_number: int, value: bool) 🔗

Based on `value`, enables or disables the specified layer in the collision_mask, given a `layer_number` between 1 and 32.

---

void **set_navigation_map**(navigation_map: RID) 🔗

Sets the RID of the navigation map this GridMap node should use for its cell baked navigation meshes.

GrooveJoint2D

Inherits: Joint2D **<** Node2D **<** CanvasItem **<** Node **<** Object

A physics joint that restricts the movement of two 2D physics bodies to a fixed axis.

Description

A physics joint that restricts the movement of two 2D physics bodies to a fixed axis. For example, a StaticBody2D representing a piston base can be attached to a RigidBody2D representing the piston head, moving up and down.

Properties

float	initial_offset	`25.0`
float	length	`50.0`

Property Descriptions

float **initial_offset** = `25.0` 🔗

- void **set_initial_offset**(value: float)
- float **get_initial_offset**()

The body B's initial anchor position defined by the joint's origin and a local offset initial_offset along the joint's Y axis (along the groove).

---

float **length** = `50.0` 🔗

- void **set_length**(value: float)
- float **get_length**()

The groove's length. The groove is from the joint's origin towards length along the joint's local Y axis.

HBoxContainer

Inherits: BoxContainer **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

Inherited By: EditorResourcePicker

A container that arranges its child controls horizontally.

Description

A variant of BoxContainer that can only arrange its child controls horizontally. Child controls are rearranged automatically when their minimum size changes.

Tutorials

- Using Containers

HFlowContainer

Inherits: FlowContainer **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

A container that arranges its child controls horizontally and wraps them around at the borders.

Description

A variant of FlowContainer that can only arrange its child controls horizontally, wrapping them around at the borders. This is similar to how text in a book wraps around when no more words can fit on a line.

Tutorials

- Using Containers

HingeJoint3D

Inherits: Joint3D **<** Node3D **<** Node **<** Object

A physics joint that restricts the rotation of a 3D physics body around an axis relative to another physics body.

Description

A physics joint that restricts the rotation of a 3D physics body around an axis relative to another physics body. For example, Body A can be a StaticBody3D representing a door hinge that a RigidBody3D rotates around.

Properties

float	angular_limit/bias	`0.3`
bool	angular_limit/enable	`false`
float	angular_limit/lower	`-1.5708`
float	angular_limit/relaxation	`1.0`
float	angular_limit/softness	`0.9`

float	angular_limit/upper	`1.5708`
bool	motor/enable	`false`
float	motor/max_impulse	`1.0`
float	motor/target_velocity	`1.0`
float	params/bias	`0.3`

Methods

bool	get_flag(flag: Flag) const
float	get_param(param: Param) const
void	set_flag(flag: Flag, enabled: bool)
void	set_param(param: Param, value: float)

Enumerations

enum **Param**: 🔗

Param **PARAM_BIAS** = `0`

The speed with which the two bodies get pulled together when they move in different directions.

Param **PARAM_LIMIT_UPPER** = `1`

The maximum rotation. Only active if angular_limit/enable is `true`.

Param **PARAM_LIMIT_LOWER** = `2`

The minimum rotation. Only active if angular_limit/enable is `true`.

Param **PARAM_LIMIT_BIAS** = `3`

The speed with which the rotation across the axis perpendicular to the hinge gets corrected.

Param **PARAM_LIMIT_SOFTNESS** = `4`

Deprecated: This property is never used by the engine and is kept for compatibility purpose.

Param **PARAM_LIMIT_RELAXATION** = `5`

The lower this value, the more the rotation gets slowed down.

Param **PARAM_MOTOR_TARGET_VELOCITY** = `6`

Target speed for the motor.

Param **PARAM_MOTOR_MAX_IMPULSE** = `7`

Maximum acceleration for the motor.

Param **PARAM_MAX** = `8`

Represents the size of the Param enum.

---

enum **Flag**: 🔗

Flag **FLAG_USE_LIMIT** = `0`

If `true`, the hinges maximum and minimum rotation, defined by angular_limit/lower and angular_limit/upper has effects.

Flag **FLAG_ENABLE_MOTOR** = `1`

When activated, a motor turns the hinge.

Flag **FLAG_MAX** = `2`

Represents the size of the Flag enum.

Property Descriptions

float **angular_limit/bias** = `0.3` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The speed with which the rotation across the axis perpendicular to the hinge gets corrected.

---

bool **angular_limit/enable** = `false` 🔗

- void **set_flag**(flag: Flag, enabled: bool)
- bool **get_flag**(flag: Flag) const

If `true`, the hinges maximum and minimum rotation, defined by angular_limit/lower and angular_limit/upper has effects.

---

float **angular_limit/lower** = `-1.5708`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The minimum rotation. Only active if angular_limit/enable is `true`.

---

float **angular_limit/relaxation** = `1.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The lower this value, the more the rotation gets slowed down.

---

float **angular_limit/softness** = `0.9`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

Deprecated: This property is never set by the engine and is kept for compatibility purposes.

---

float **angular_limit/upper** = `1.5708`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The maximum rotation. Only active if angular_limit/enable is `true`.

---

bool **motor/enable** = `false`

- void **set_flag**(flag: Flag, enabled: bool)
- bool **get_flag**(flag: Flag) const

When activated, a motor turns the hinge.

---

float **motor/max_impulse** = `1.0` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

Maximum acceleration for the motor.

---

float **motor/target_velocity** = `1.0` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

Target speed for the motor.

---

float **params/bias** = `0.3` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The speed with which the two bodies get pulled together when they move in different directions.

Method Descriptions

bool **get_flag**(flag: Flag) const 🔗

Returns the value of the specified flag.

---

float **get_param**(param: Param) const 🔗

Returns the value of the specified parameter.

void **set_flag**(flag: Flag, enabled: bool) 🔗

If `true`, enables the specified flag.

void **set_param**(param: Param, value: float) 🔗

Sets the value of the specified parameter.

HScrollBar

Inherits: ScrollBar **<** Range **<** Control **<** CanvasItem **<** Node **<** Object

A horizontal scrollbar that goes from left (min) to right (max).

Description

A horizontal scrollbar, typically used to navigate through content that extends beyond the visible width of a control. It is a Range-based control and goes from left (min) to right (max).

HSeparator

Inherits: Separator **<** Control **<** CanvasItem **<** Node **<** Object

A horizontal line used for separating other controls.

Description

A horizontal separator used for separating other controls that are arranged **vertically**. **HSeparator** is purely visual and normally drawn as a StyleBoxLine.

HSlider

Inherits: Slider **<** Range **<** Control **<** CanvasItem **<** Node **<** Object

A horizontal slider that goes from left (min) to right (max).

Description

A horizontal slider, used to adjust a value by moving a grabber along a horizontal axis. It is a Range-based control and goes from left (min) to right (max).

HSplitContainer

Inherits: SplitContainer **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

A container that splits two child controls horizontally and provides a grabber for adjusting the split ratio.

Description

A container that accepts only two child controls, then arranges them horizontally and creates a divisor between them. The divisor can be dragged around to change the size relation between the child controls.

Tutorials

- Using Containers

HTTPRequest

Inherits: Node < Object

A node with the ability to send HTTP(S) requests.

Description

A node with the ability to send HTTP requests. Uses HTTPClient internally.

Can be used to make HTTP requests, i.e. download or upload files or web content via HTTP.

Warning: See the notes and warnings on HTTPClient for limitations, especially regarding TLS security.

Note: When exporting to Android, make sure to enable the `INTERNET` permission in the Android export preset before exporting the project or using one-click deploy. Otherwise, network communication of any kind will be blocked by Android.

Example of contacting a REST API and printing one of its returned fields:

GDScript

```
func _ready():
    # Create an HTTP request node and connect its completion
signal.
    var http_request = HTTPRequest.new()
    add_child(http_request)

http_request.request_completed.connect(self._http_request_compl
eted)
```

```
    # Perform a GET request. The URL below returns JSON as of
writing.
    var error = http_request.request("https://httpbin.org/get")
    if error != OK:
        push_error("An error occurred in the HTTP request.")

    # Perform a POST request. The URL below returns JSON as of
writing.
    # Note: Don't make simultaneous requests using a single
HTTPRequest node.
    # The snippet below is provided for reference only.
    var body = JSON.new().stringify({"name": "Godette"})
    error = http_request.request("https://httpbin.org/post",
[], HTTPClient.METHOD_POST, body)
    if error != OK:
        push_error("An error occurred in the HTTP request.")

# Called when the HTTP request is completed.
func _http_request_completed(result, response_code, headers,
body):
    var json = JSON.new()
    json.parse(body.get_string_from_utf8())
    var response = json.get_data()

    # Will print the user agent string used by the HTTPRequest
node (as recognized by httpbin.org).
    print(response.headers["User-Agent"])
```

C#

```
public override void _Ready()
{
    // Create an HTTP request node and connect its completion
signal.
    var httpRequest = new HttpRequest();
    AddChild(httpRequest);
    httpRequest.RequestCompleted += HttpRequestCompleted;

    // Perform a GET request. The URL below returns JSON as of
writing.
    Error error =
httpRequest.Request("https://httpbin.org/get");
    if (error != Error.Ok)
    {
        GD.PushError("An error occurred in the HTTP request.");
    }

    // Perform a POST request. The URL below returns JSON as of
writing.
    // Note: Don't make simultaneous requests using a single
HTTPRequest node.
    // The snippet below is provided for reference only.
    string body = new Json().Stringify(new
Godot.Collections.Dictionary
    {
        { "name", "Godette" }
    });
    error = httpRequest.Request("https://httpbin.org/post",
null, HttpClient.Method.Post, body);
    if (error != Error.Ok)
    {
        GD.PushError("An error occurred in the HTTP request.");
    }
}

// Called when the HTTP request is completed.
private void HttpRequestCompleted(long result, long
responseCode, string[] headers, byte[] body)
{
    var json = new Json();
    json.Parse(body.GetStringFromUtf8());
    var response = json.GetData().AsGodotDictionary();

    // Will print the user agent string used by the HTTPRequest
node (as recognized by httpbin.org).
    GD.Print((response["headers"].AsGodotDictionary())["User-
Agent"]);
}
```

Example of loading and displaying an image using HTTPRequest:

GDScript

```
func _ready():
    # Create an HTTP request node and connect its completion
signal.
    var http_request = HTTPRequest.new()
    add_child(http_request)

http_request.request_completed.connect(self._http_request_compl
eted)

    # Perform the HTTP request. The URL below returns a PNG
image as of writing.
    var error =
http_request.request("https://via.placeholder.com/512")
    if error != OK:
        push_error("An error occurred in the HTTP request.")

# Called when the HTTP request is completed.
func _http_request_completed(result, response_code, headers,
body):
    if result != HTTPRequest.RESULT_SUCCESS:
        push_error("Image couldn't be downloaded. Try a
different image.")

    var image = Image.new()
    var error = image.load_png_from_buffer(body)
    if error != OK:
        push_error("Couldn't load the image.")

    var texture = ImageTexture.create_from_image(image)

    # Display the image in a TextureRect node.
    var texture_rect = TextureRect.new()
    add_child(texture_rect)
    texture_rect.texture = texture
```

C#

```
public override void _Ready()
{
    // Create an HTTP request node and connect its completion
signal.
    var httpRequest = new HttpRequest();
    AddChild(httpRequest);
    httpRequest.RequestCompleted += HttpRequestCompleted;

    // Perform the HTTP request. The URL below returns a PNG
image as of writing.
    Error error =
httpRequest.Request("https://via.placeholder.com/512");
    if (error != Error.Ok)
    {
        GD.PushError("An error occurred in the HTTP request.");
    }
}

// Called when the HTTP request is completed.
private void HttpRequestCompleted(long result, long
responseCode, string[] headers, byte[] body)
{
    if (result != (long)HttpRequest.Result.Success)
    {
        GD.PushError("Image couldn't be downloaded. Try a
different image.");
    }
    var image = new Image();
    Error error = image.LoadPngFromBuffer(body);
    if (error != Error.Ok)
    {
        GD.PushError("Couldn't load the image.");
    }

    var texture = ImageTexture.CreateFromImage(image);

    // Display the image in a TextureRect node.
    var textureRect = new TextureRect();
    AddChild(textureRect);
    textureRect.Texture = texture;
}
```

Gzipped response bodies: HTTPRequest will automatically handle decompression of response bodies. A `Accept-Encoding` header will be automatically added to each of your requests, unless one is already specified. Any response with a `Content-Encoding: gzip` header will automatically be decompressed and delivered to you as uncompressed bytes.

Tutorials

- Making HTTP requests
- TLS certificates

Properties

bool	accept_gzip	`true`
int	body_size_limit	`-1`
int	download_chunk_size	`65536`
String	download_file	`""`
int	max_redirects	`8`
float	timeout	`0.0`
bool	use_threads	`false`

Methods

void	cancel_request()
int	get_body_size() const
int	get_downloaded_bytes() const
Status	get_http_client_status() const
Error	request(url: String, custom_headers: PackedStringArray = PackedStringArray(), method: Method = 0, request_data: String = "")
Error	request_raw(url: String, custom_headers: PackedStringArray = PackedStringArray(), method: Method = 0, request_data_raw: PackedByteArray = PackedByteArray())
void	set_http_proxy(host: String, port: int)
void	set_https_proxy(host: String, port: int)
void	set_tls_options(client_options: TLSOptions)

Signals

request_completed(result: int, response_code: int, headers: PackedStringArray, body: PackedByteArray) 🔗

Emitted when a request is completed.

Enumerations

enum **Result**: 🔗

Result **RESULT_SUCCESS** = `0`

Request successful.

Result **RESULT_CHUNKED_BODY_SIZE_MISMATCH** = `1`

There is currently no description for this enum. Please help us by contributing one!

Result **RESULT_CANT_CONNECT** = `2`

Request failed while connecting.

Result **RESULT_CANT_RESOLVE** = `3`

Request failed while resolving.

Result **RESULT_CONNECTION_ERROR** = `4`

Request failed due to connection (read/write) error.

Result **RESULT_TLS_HANDSHAKE_ERROR** = `5`

Request failed on TLS handshake.

Result **RESULT_NO_RESPONSE** = `6`

Request does not have a response (yet).

Result **RESULT_BODY_SIZE_LIMIT_EXCEEDED** = `7`

Request exceeded its maximum size limit, see body_size_limit.

Result **RESULT_BODY_DECOMPRESS_FAILED** = `8`

There is currently no description for this enum. Please help us by contributing one!

Result **RESULT_REQUEST_FAILED** = `9`

Request failed (currently unused).

Result **RESULT_DOWNLOAD_FILE_CANT_OPEN** = `10`

HTTPRequest couldn't open the download file.

Result **RESULT_DOWNLOAD_FILE_WRITE_ERROR** = `11`

HTTPRequest couldn't write to the download file.

Result **RESULT_REDIRECT_LIMIT_REACHED** = `12`

Request reached its maximum redirect limit, see max_redirects.

Result **RESULT_TIMEOUT** = `13`

Request failed due to a timeout. If you expect requests to take a long time, try increasing the value of timeout or setting it to `0.0` to remove the timeout completely.

Property Descriptions

bool **accept_gzip** = `true` 🔗

- void **set_accept_gzip**(value: bool)
- bool **is_accepting_gzip**()

If `true`, this header will be added to each request: `Accept-Encoding: gzip, deflate` telling servers that it's okay to compress response bodies.

Any Response body declaring a `Content-Encoding` of either `gzip` or `deflate` will then be automatically decompressed, and the uncompressed bytes will be delivered via request_completed.

If the user has specified their own `Accept-Encoding` header, then no header will be added regardless of accept_gzip.

If `false` no header will be added, and no decompression will be performed on response bodies. The raw bytes of the response body will be returned via request_completed.

---

int **body_size_limit** = `-1` 🔗

- void **set_body_size_limit**(value: int)
- int **get_body_size_limit**()

Maximum allowed size for response bodies. If the response body is compressed, this will be used as the maximum allowed size for the decompressed body.

---

int **download_chunk_size** = `65536` 🔗

- void **set_download_chunk_size**(value: int)

- int **get_download_chunk_size**()

The size of the buffer used and maximum bytes to read per iteration. See HTTPClient.read_chunk_size.

Set this to a lower value (e.g. 4096 for 4 KiB) when downloading small files to decrease memory usage at the cost of download speeds.

String **download_file** = `""`

- void **set_download_file**(value: String)
- String **get_download_file**()

The file to download into. Will output any received file into it.

---

int **max_redirects** = `8`

- void **set_max_redirects**(value: int)
- int **get_max_redirects**()

Maximum number of allowed redirects.

---

float **timeout** = `0.0`

- void **set_timeout**(value: float)
- float **get_timeout**()

The duration to wait in seconds before a request times out. If timeout is set to `0.0` then the request will never time out. For simple requests, such as communication with a REST API, it is recommended that timeout is set to a value suitable for the server response time (e.g. between `1.0` and `10.0`). This will help prevent unwanted timeouts caused by variation in server response times while still allowing the application to detect when a request has timed out. For larger requests such as file downloads it is suggested the timeout be set to `0.0`, disabling the timeout functionality. This will help to prevent large transfers from failing due to exceeding the timeout value.

bool **use_threads** = `false`

- void **set_use_threads**(value: bool)
- bool **is_using_threads**()

If `true`, multithreading is used to improve performance.

Method Descriptions

void **cancel_request**()

Cancels the current request.

int **get_body_size**() const

Returns the response body length.

Note: Some Web servers may not send a body length. In this case, the value returned will be `-1`. If using chunked transfer encoding, the body length will also be `-1`.

int **get_downloaded_bytes**() const

Returns the number of bytes this HTTPRequest downloaded.

Status **get_http_client_status**() const 🔗

Returns the current status of the underlying HTTPClient. See Status.

Error **request**(url: String, custom_headers: PackedStringArray = PackedStringArray(), method: Method = 0, request_data: String = "") 🔗

Creates request on the underlying HTTPClient. If there is no configuration errors, it tries to connect using HTTPClient.connect_to_host and passes parameters onto HTTPClient.request.

Returns @GlobalScope.OK if request is successfully created. (Does not imply that the server has responded), @GlobalScope.ERR_UNCONFIGURED if not in the tree, @GlobalScope.ERR_BUSY if still processing previous request, @GlobalScope.ERR_INVALID_PARAMETER if given string is not a valid URL format, or @GlobalScope.ERR_CANT_CONNECT if not using thread and the HTTPClient cannot connect to host.

Note: When `method` is HTTPClient.METHOD_GET, the payload sent via `request_data` might be ignored by the server or even cause the server to reject the request (check RFC 7231 section 4.3.1 [https://datatracker.ietf.org/doc/html/rfc7231#section-4.3.1] for more details). As a workaround, you can send data as a query string in the URL (see String.uri_encode for an example).

Note: It's recommended to use transport encryption (TLS) and to avoid sending sensitive information (such as login credentials) in HTTP GET URL parameters. Consider using HTTP POST requests or HTTP headers for such information instead.

Error **request_raw**(url: String, custom_headers: PackedStringArray = PackedStringArray(), method: Method = 0, request_data_raw: PackedByteArray = PackedByteArray()) 🔗

Creates request on the underlying HTTPClient using a raw array of bytes for the request body. If there is no configuration errors, it tries to connect using HTTPClient.connect_to_host and passes parameters onto HTTPClient.request.

Returns @GlobalScope.OK if request is successfully created. (Does not imply that the server has responded), @GlobalScope.ERR_UNCONFIGURED if not in the tree, @GlobalScope.ERR_BUSY if still processing previous request, @GlobalScope.ERR_INVALID_PARAMETER if given string is not a valid URL format, or @GlobalScope.ERR_CANT_CONNECT if not using thread and the HTTPClient cannot connect to host.

---

void **set_http_proxy**(host: String, port: int) 🔗

Sets the proxy server for HTTP requests.

The proxy server is unset if `host` is empty or `port` is -1.

---

void **set_https_proxy**(host: String, port: int) 🔗

Sets the proxy server for HTTPS requests.

The proxy server is unset if `host` is empty or `port` is -1.

---

void **set_tls_options**(client_options: TLSOptions) 🔗

Sets the TLSOptions to be used when connecting to an HTTPS server. See TLSOptions.client.

ImporterMeshInstance3D

Inherits: Node3D **<** Node **<** Object

There is currently no description for this class. Please help us by contributing one!

Properties

ShadowCastingSetting	cast_shadow	`1`
int	layer_mask	`1`
ImporterMesh	mesh	
NodePath	skeleton_path	`NodePath("")`
Skin	skin	
float	visibility_range_begin	`0.0`
float	visibility_range_begin_margin	`0.0`
float	visibility_range_end	`0.0`
float	visibility_range_end_margin	`0.0`

VisibilityRangeFadeMode	visibility_range_fade_mode	`0`

Property Descriptions

ShadowCastingSetting **cast_shadow** = `1` 🔗

- void **set_cast_shadows_setting**(value: ShadowCastingSetting)
- ShadowCastingSetting **get_cast_shadows_setting**()

There is currently no description for this property. Please help us by contributing one!

---

int **layer_mask** = `1` 🔗

- void **set_layer_mask**(value: int)
- int **get_layer_mask**()

There is currently no description for this property. Please help us by contributing one!

ImporterMesh **mesh** 🔗

- void **set_mesh**(value: ImporterMesh)
- ImporterMesh **get_mesh**()

There is currently no description for this property. Please help us by contributing one!

NodePath **skeleton_path** = `NodePath("")` 🔗

- void **set_skeleton_path**(value: NodePath)

- NodePath **get_skeleton_path**()

There is currently no description for this property. Please help us by contributing one!

Skin **skin** 🔗

- void **set_skin**(value: Skin)
- Skin **get_skin**()

There is currently no description for this property. Please help us by contributing one!

float **visibility_range_begin** = `0.0` 🔗

- void **set_visibility_range_begin**(value: float)
- float **get_visibility_range_begin**()

There is currently no description for this property. Please help us by contributing one!

float **visibility_range_begin_margin** = `0.0` 🔗

- void **set_visibility_range_begin_margin**(value: float)
- float **get_visibility_range_begin_margin**()

There is currently no description for this property. Please help us by contributing one!

float **visibility_range_end** = `0.0` 🔗

- void **set_visibility_range_end**(value: float)
- float **get_visibility_range_end**()

There is currently no description for this property. Please help us by contributing one!

---

float **visibility_range_end_margin** = `0.0` 🔗

- void **set_visibility_range_end_margin**(value: float)
- float **get_visibility_range_end_margin**()

There is currently no description for this property. Please help us by contributing one!

VisibilityRangeFadeMode **visibility_range_fade_mode** = `0` 🔗

- void **set_visibility_range_fade_mode**(value: VisibilityRangeFadeMode)
- VisibilityRangeFadeMode **get_visibility_range_fade_mode**()

There is currently no description for this property. Please help us by contributing one!

InstancePlaceholder

Inherits: Node **<** Object

Placeholder for the root Node of a PackedScene.

Description

Turning on the option **Load As Placeholder** for an instantiated scene in the editor causes it to be replaced by an **InstancePlaceholder** when running the game, this will not replace the node in the editor. This makes it possible to delay actually loading the scene until calling create_instance. This is useful to avoid loading large scenes all at once by loading parts of it selectively.

The **InstancePlaceholder** does not have a transform. This causes any child nodes to be positioned relatively to the Viewport from point (0,0), rather than their parent as displayed in the editor. Replacing the placeholder with a scene with a transform will transform children relatively to their parent again.

Methods

Node	create_instance(replace: bool = false, custom_scene: PackedScene = null)
String	get_instance_path() const
Dictionary	get_stored_values(with_order: bool = false)

Method Descriptions

Node **create_instance**(replace: bool = false, custom_scene: PackedScene = null) 🔗

Call this method to actually load in the node. The created node will be placed as a sibling *above* the **InstancePlaceholder** in the scene tree. The Node's reference is also returned for convenience.

Note: create_instance is not thread-safe. Use Object.call_deferred if calling from a thread.

String **get_instance_path**() const 🔗

Gets the path to the PackedScene resource file that is loaded by default when calling create_instance. Not thread-safe. Use Object.call_deferred if calling from a thread.

Dictionary **get_stored_values**(with_order: bool = false) 🔗

Returns the list of properties that will be applied to the node when create_instance is called.

If `with_order` is `true`, a key named `.order` (note the leading period) is added to the dictionary. This `.order` key is an Array of String property names specifying the order in which properties will be applied (with index 0 being the first).

ItemList

Inherits: Control **<** CanvasItem **<** Node **<** Object

A vertical list of selectable items with one or multiple columns.

Description

This control provides a vertical list of selectable items that may be in a single or in multiple columns, with each item having options for text and an icon. Tooltips are supported and may be different for every item in the list.

Selectable items in the list may be selected or deselected and multiple selection may be enabled. Selection with right mouse button may also be enabled to allow use of popup context menus. Items may also be "activated" by double-clicking them or by pressing `Enter`.

Item text only supports single-line strings. Newline characters (e.g. `\n`) in the string won't produce a newline. Text wrapping is enabled in ICON_MODE_TOP mode, but the column's width is adjusted to fully fit its content by default. You need to set fixed_column_width greater than zero to wrap the text.

All `set_*` methods allow negative item indices, i.e. `-1` to access the last item, `-2` to select the second-to-last item, and so on.

Incremental search: Like PopupMenu and Tree, **ItemList** supports searching within the list while the control is focused. Press a key that matches the first letter of an item's name to select the first item starting with the given letter. After that point, there are two ways to perform incremental search: 1) Press the same key again before the timeout duration to select the next item starting with the same letter. 2) Press letter keys that match the rest of the word before the timeout duration to match to select the item in question directly. Both of these actions will be reset to the beginning of the list if the timeout duration has passed since the last keystroke was registered. You can adjust the timeout duration by changing ProjectSettings.gui/timers/incremental_search_max_interval_msec.

Properties

bool	allow_reselect	`false`
bool	allow_rmb_select	`false`
bool	allow_search	`true`
bool	auto_height	`false`
bool	clip_contents	`true` (overrides Control)
int	fixed_column_width	`0`
Vector2i	fixed_icon_size	`Vector2i(0, 0)`
FocusMode	focus_mode	`2` (overrides Control)
IconMode	icon_mode	`1`

float	icon_scale	`1.0`
int	item_count	`0`
int	max_columns	`1`
int	max_text_lines	`1`
bool	same_column_width	`false`
SelectMode	select_mode	`0`
OverrunBehavior	text_overrun_behavior	`3`

Methods

int	add_icon_item(icon: Texture2D, selectable: bool = true)
int	add_item(text: String, icon: Texture2D = null, selectable: bool = true)
void	clear()

void	deselect(idx: int)
void	deselect_all()
void	ensure_current_is_visible()
void	force_update_list_size()
int	get_item_at_position(position: Vector2, exact: bool = false) const
Color	get_item_custom_bg_color(idx: int) const
Color	get_item_custom_fg_color(idx: int) const
Texture2D	get_item_icon(idx: int) const
Color	get_item_icon_modulate(idx: int) const
Rect2	get_item_icon_region(idx: int) const
String	get_item_language(idx: int) const
Variant	get_item_metadata(idx: int) const

Rect2	get_item_rect(idx: int, expand: bool = true) const
String	get_item_text(idx: int) const
TextDirection	get_item_text_direction(idx: int) const
String	get_item_tooltip(idx: int) const
PackedInt32Array	get_selected_items()
VScrollBar	get_v_scroll_bar()
bool	is_anything_selected()
bool	is_item_disabled(idx: int) const
bool	is_item_icon_transposed(idx: int) const
bool	is_item_selectable(idx: int) const
bool	is_item_tooltip_enabled(idx: int) const
bool	is_selected(idx: int) const

void	move_item(from_idx: int, to_idx: int)
void	remove_item(idx: int)
void	select(idx: int, single: bool = true)
void	set_item_custom_bg_color(idx: int, custom_bg_color: Color)
void	set_item_custom_fg_color(idx: int, custom_fg_color: Color)
void	set_item_disabled(idx: int, disabled: bool)
void	set_item_icon(idx: int, icon: Texture2D)
void	set_item_icon_modulate(idx: int, modulate: Color)
void	set_item_icon_region(idx: int, rect: Rect2)
void	set_item_icon_transposed(idx: int, transposed: bool)
void	set_item_language(idx: int, language: String)

void	set_item_metadata(idx: int, metadata: Variant)
void	set_item_selectable(idx: int, selectable: bool)
void	set_item_text(idx: int, text: String)
void	set_item_text_direction(idx: int, direction: TextDirection)
void	set_item_tooltip(idx: int, tooltip: String)
void	set_item_tooltip_enabled(idx: int, enable: bool)
void	sort_items_by_text()

Theme Properties

Color	font_color	`Color(0.65, 0.65, 0.65, 1)`
Color	font_hovered_color	`Color(0.95, 0.95, 0.95, 1)`
Color	font_outline_color	`Color(0, 0, 0, 1)`
Color	font_selected_color	`Color(1, 1, 1, 1)`

Color	guide_color	`Color(0.7, 0.7, 0.7, 0.25)`
int	h_separation	`4`
int	icon_margin	`4`
int	line_separation	`2`
int	outline_size	`0`
int	v_separation	`4`
Font	font	
int	font_size	
StyleBox	cursor	
StyleBox	cursor_unfocused	
StyleBox	focus	
StyleBox	hovered	
StyleBox	panel	

StyleBox selected

StyleBox selected_focus

Signals

empty_clicked(at_position: Vector2, mouse_button_index: int) 🔗

Triggered when any mouse click is issued within the rect of the list but on empty space.

item_activated(index: int) 🔗

Triggered when specified list item is activated via double-clicking or by pressing `Enter`.

item_clicked(index: int, at_position: Vector2, mouse_button_index: int) 🔗

Triggered when specified list item has been clicked with any mouse button.

The click position is also provided to allow appropriate popup of context menus at the correct location.

item_selected(index: int) 🔗

Triggered when specified item has been selected.

allow_reselect must be enabled to reselect an item.

---

multi_selected(index: int, selected: bool) 🔗

Triggered when a multiple selection is altered on a list allowing multiple selection.

Enumerations

enum **IconMode**: 🔗

IconMode **ICON_MODE_TOP** = `0`

Icon is drawn above the text.

IconMode **ICON_MODE_LEFT** = `1`

Icon is drawn to the left of the text.

---

enum **SelectMode**: 🔗

SelectMode **SELECT_SINGLE** = `0`

Only allow selecting a single item.

SelectMode **SELECT_MULTI** = `1`

Allows selecting multiple items by holding `Ctrl` or `Shift`.

Property Descriptions

bool **allow_reselect** = `false` 🔗

- void **set_allow_reselect**(value: bool)
- bool **get_allow_reselect**()

If `true`, the currently selected item can be selected again.

---

bool **allow_rmb_select** = `false` 🔗

- void **set_allow_rmb_select**(value: bool)
- bool **get_allow_rmb_select**()

If `true`, right mouse button click can select items.

---

bool **allow_search** = `true` 🔗

- void **set_allow_search**(value: bool)
- bool **get_allow_search**()

If `true`, allows navigating the **ItemList** with letter keys through incremental search.

---

bool **auto_height** = `false` 🔗

- void **set_auto_height**(value: bool)
- bool **has_auto_height**()

If `true`, the control will automatically resize the height to fit its content.

---

int **fixed_column_width** = `0` 🔗

- void **set_fixed_column_width**(value: int)

- int **get_fixed_column_width**()

The width all columns will be adjusted to.

A value of zero disables the adjustment, each item will have a width equal to the width of its content and the columns will have an uneven width.

Vector2i **fixed_icon_size** = `Vector2i(0, 0)`

- void **set_fixed_icon_size**(value: Vector2i)
- Vector2i **get_fixed_icon_size**()

The size all icons will be adjusted to.

If either X or Y component is not greater than zero, icon size won't be affected.

IconMode **icon_mode** = `1`

- void **set_icon_mode**(value: IconMode)
- IconMode **get_icon_mode**()

The icon position, whether above or to the left of the text. See the IconMode constants.

---

float **icon_scale** = `1.0`

- void **set_icon_scale**(value: float)
- float **get_icon_scale**()

The scale of icon applied after fixed_icon_size and transposing takes effect.

---

int **item_count** = `0` 🔗

- void **set_item_count**(value: int)
- int **get_item_count**()

The number of items currently in the list.

int **max_columns** = `1` 🔗

- void **set_max_columns**(value: int)
- int **get_max_columns**()

Maximum columns the list will have.

If greater than zero, the content will be split among the specified columns.

A value of zero means unlimited columns, i.e. all items will be put in the same row.

int **max_text_lines** = `1` 🔗

- void **set_max_text_lines**(value: int)
- int **get_max_text_lines**()

Maximum lines of text allowed in each item. Space will be reserved even when there is not enough lines of text to display.

Note: This property takes effect only when icon_mode is ICON_MODE_TOP. To make the text wrap, fixed_column_width should be greater than zero.

bool **same_column_width** = `false` 🔗

- void **set_same_column_width**(value: bool)
- bool **is_same_column_width**()

Whether all columns will have the same width.

If `true`, the width is equal to the largest column width of all columns.

SelectMode **select_mode** = `0` 🔗

- void **set_select_mode**(value: SelectMode)
- SelectMode **get_select_mode**()

Allows single or multiple item selection. See the SelectMode constants.

OverrunBehavior **text_overrun_behavior** = `3` 🔗

- void **set_text_overrun_behavior**(value: OverrunBehavior)
- OverrunBehavior **get_text_overrun_behavior**()

Sets the clipping behavior when the text exceeds an item's bounding rectangle. See OverrunBehavior for a description of all modes.

Method Descriptions

int **add_icon_item**(icon: Texture2D, selectable: bool = true) 🔗

Adds an item to the item list with no text, only an icon. Returns the index of an added item.

---

int **add_item**(text: String, icon: Texture2D = null, selectable: bool = true) 🔗

Adds an item to the item list with specified text. Returns the index of an added item.

Specify an `icon`, or use `null` as the `icon` for a list item with no icon.

If selectable is `true`, the list item will be selectable.

---

void **clear**()

Removes all items from the list.

---

void **deselect**(idx: int)

Ensures the item associated with the specified index is not selected.

---

void **deselect_all**()

Ensures there are no items selected.

---

void **ensure_current_is_visible**()

Ensure current selection is visible, adjusting the scroll position as necessary.

---

void **force_update_list_size**()

Forces an update to the list size based on its items. This happens automatically whenever size of the items, or other relevant settings like auto_height, change. The method can be used to trigger the update ahead of next drawing pass.

---

int **get_item_at_position**(position: Vector2, exact: bool = false) const

Returns the item index at the given `position`.

When there is no item at that point, -1 will be returned if `exact` is `true`, and the closest item index will be returned otherwise.

Note: The returned value is unreliable if called right after modifying the **ItemList**, before it redraws in the next frame.

Color **get_item_custom_bg_color**(idx: int) const

Returns the custom background color of the item specified by `idx` index.

Color **get_item_custom_fg_color**(idx: int) const

Returns the custom foreground color of the item specified by `idx` index.

Texture2D **get_item_icon**(idx: int) const

Returns the icon associated with the specified index.

Color **get_item_icon_modulate**(idx: int) const

Returns a Color modulating item's icon at the specified index.

Rect2 **get_item_icon_region**(idx: int) const

Returns the region of item's icon used. The whole icon will be used if the region has no area.

String **get_item_language**(idx: int) const 🔗

Returns item's text language code.

Variant **get_item_metadata**(idx: int) const 🔗

Returns the metadata value of the specified index.

Rect2 **get_item_rect**(idx: int, expand: bool = true) const 🔗

Returns the position and size of the item with the specified index, in the coordinate system of the **ItemList** node. If `expand` is `true` the last column expands to fill the rest of the row.

Note: The returned value is unreliable if called right after modifying the **ItemList**, before it redraws in the next frame.

String **get_item_text**(idx: int) const 🔗

Returns the text associated with the specified index.

TextDirection **get_item_text_direction**(idx: int) const 🔗

Returns item's text base writing direction.

String **get_item_tooltip**(idx: int) const 🔗

Returns the tooltip hint associated with the specified index.

PackedInt32Array **get_selected_items**()

Returns an array with the indexes of the selected items.

VScrollBar **get_v_scroll_bar**()

Returns the vertical scrollbar.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their CanvasItem.visible property.

bool **is_anything_selected**()

Returns `true` if one or more items are selected.

bool **is_item_disabled**(idx: int) const

Returns `true` if the item at the specified index is disabled.

bool **is_item_icon_transposed**(idx: int) const

Returns `true` if the item icon will be drawn transposed, i.e. the X and Y axes are swapped.

bool **is_item_selectable**(idx: int) const

Returns `true` if the item at the specified index is selectable.

bool **is_item_tooltip_enabled**(idx: int) const

Returns `true` if the tooltip is enabled for specified item index.

---

bool **is_selected**(idx: int) const

Returns `true` if the item at the specified index is currently selected.

---

void **move_item**(from_idx: int, to_idx: int)

Moves item from index `from_idx` to `to_idx`.

---

void **remove_item**(idx: int)

Removes the item specified by `idx` index from the list.

---

void **select**(idx: int, single: bool = true)

Select the item at the specified index.

Note: This method does not trigger the item selection signal.

---

void **set_item_custom_bg_color**(idx: int, custom_bg_color: Color)

Sets the background color of the item specified by `idx` index to the specified Color.

---

void **set_item_custom_fg_color**(idx: int, custom_fg_color: Color)

Sets the foreground color of the item specified by `idx` index to the specified Color.

void **set_item_disabled**(idx: int, disabled: bool)

Disables (or enables) the item at the specified index.

Disabled items cannot be selected and do not trigger activation signals (when double-clicking or pressing `Enter`).

void **set_item_icon**(idx: int, icon: Texture2D)

Sets (or replaces) the icon's Texture2D associated with the specified index.

void **set_item_icon_modulate**(idx: int, modulate: Color)

Sets a modulating Color of the item associated with the specified index.

void **set_item_icon_region**(idx: int, rect: Rect2)

Sets the region of item's icon used. The whole icon will be used if the region has no area.

void **set_item_icon_transposed**(idx: int, transposed: bool)

Sets whether the item icon will be drawn transposed.

void **set_item_language**(idx: int, language: String)

Sets language code of item's text used for line-breaking and text shaping algorithms, if left empty current locale is used instead.

void **set_item_metadata**(idx: int, metadata: Variant)

Sets a value (of any type) to be stored with the item associated with the specified index.

void **set_item_selectable**(idx: int, selectable: bool)

Allows or disallows selection of the item associated with the specified index.

void **set_item_text**(idx: int, text: String)

Sets text of the item associated with the specified index.

void **set_item_text_direction**(idx: int, direction: TextDirection)

Sets item's text base writing direction.

void **set_item_tooltip**(idx: int, tooltip: String)

Sets the tooltip hint for the item associated with the specified index.

void **set_item_tooltip_enabled**(idx: int, enable: bool)

Sets whether the tooltip hint is enabled for specified item index.

void **sort_items_by_text**()

Sorts items in the list by their text.

Theme Property Descriptions

Color **font_color** = `Color(0.65, 0.65, 0.65, 1)`

Default text Color of the item.

Color **font_hovered_color** = `Color(0.95, 0.95, 0.95, 1)`

Text Color used when the item is hovered and not selected yet.

Color **font_outline_color** = `Color(0, 0, 0, 1)`

The tint of text outline of the item.

Color **font_selected_color** = `Color(1, 1, 1, 1)`

Text Color used when the item is selected.

Color **guide_color** = `Color(0.7, 0.7, 0.7, 0.25)`

Color of the guideline. The guideline is a line drawn between each row of items.

int **h_separation** = `4`

The horizontal spacing between items.

int **icon_margin** = `4`

The spacing between item's icon and text.

---

int **line_separation** = `2`

The vertical spacing between each line of text.

---

int **outline_size** = `0`

The size of the item text outline.

Note: If using a font with FontFile.multichannel_signed_distance_field enabled, its FontFile.msdf_pixel_range must be set to at least *twice* the value of outline_size for outline rendering to look correct. Otherwise, the outline may appear to be cut off earlier than intended.

---

int **v_separation** = `4`

The vertical spacing between items.

Font **font**

Font of the item's text.

---

int **font_size**

Font size of the item's text.

StyleBox **cursor**

StyleBox used for the cursor, when the **ItemList** is being focused.

StyleBox **cursor_unfocused** 🔗

StyleBox used for the cursor, when the **ItemList** is not being focused.

StyleBox **focus** 🔗

The focused style for the **ItemList**, drawn on top of the background, but below everything else.

StyleBox **hovered** 🔗

StyleBox for the hovered, but not selected items.

StyleBox **panel** 🔗

The background style for the **ItemList**.

StyleBox **selected** 🔗

StyleBox for the selected items, used when the **ItemList** is not being focused.

StyleBox **selected_focus** 🔗

StyleBox for the selected items, used when the **ItemList** is being focused.

Joint2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Inherited By: DampedSpringJoint2D, GrooveJoint2D, PinJoint2D

Abstract base class for all 2D physics joints.

Description

Abstract base class for all joints in 2D physics. 2D joints bind together two physics bodies (node_a and node_b) and apply a constraint.

Properties

float	bias	`0.0`
bool	disable_collision	`true`
NodePath	node_a	`NodePath("")`
NodePath	node_b	`NodePath("")`

Methods

RID get_rid() const

Property Descriptions

float **bias** = `0.0` 🔗

- void **set_bias**(value: float)
- float **get_bias**()

When node_a and node_b move in different directions the bias controls how fast the joint pulls them back to their original position. The lower the bias the more the two bodies can pull on the joint.

When set to `0`, the default value from ProjectSettings.physics/2d/solver/default_constraint_bias is used.

---

bool **disable_collision** = `true` 🔗

- void **set_exclude_nodes_from_collision**(value: bool)
- bool **get_exclude_nodes_from_collision**()

If `true`, the two bodies bound together do not collide with each other.

NodePath **node_a** = `NodePath("")` 🔗

- void **set_node_a**(value: NodePath)
- NodePath **get_node_a**()

Path to the first body (A) attached to the joint. The node must inherit PhysicsBody2D.

NodePath **node_b** = `NodePath("")`

- void **set_node_b**(value: NodePath)
- NodePath **get_node_b**()

Path to the second body (B) attached to the joint. The node must inherit PhysicsBody2D.

Method Descriptions

RID **get_rid**() const

Returns the joint's internal RID from the PhysicsServer2D.

Joint3D

Inherits: Node3D **<** Node **<** Object

Inherited By: ConeTwistJoint3D, Generic6DOFJoint3D, HingeJoint3D, PinJoint3D, SliderJoint3D

Abstract base class for all 3D physics joints.

Description

Abstract base class for all joints in 3D physics. 3D joints bind together two physics bodies (node_a and node_b) and apply a constraint. If only one body is defined, it is attached to a fixed StaticBody3D without collision shapes.

Tutorials

- 3D Truck Town Demo [https://godotengine.org/asset-library/asset/2752]

Properties

bool	exclude_nodes_from_collision	`true`
NodePath	node_a	`NodePath("")`
NodePath	node_b	`NodePath("")`

int	solver_priority	`1`

Methods

RID	get_rid() const

Property Descriptions

bool **exclude_nodes_from_collision** = `true`

- void **set_exclude_nodes_from_collision**(value: bool)
- bool **get_exclude_nodes_from_collision**()

If `true`, the two bodies bound together do not collide with each other.

NodePath **node_a** = `NodePath("")`

- void **set_node_a**(value: NodePath)
- NodePath **get_node_a**()

Path to the first node (A) attached to the joint. The node must inherit PhysicsBody3D.

If left empty and node_b is set, the body is attached to a fixed StaticBody3D without collision shapes.

NodePath **node_b** = `NodePath("")`

- void **set_node_b**(value: NodePath)
- NodePath **get_node_b**()

Path to the second node (B) attached to the joint. The node must inherit PhysicsBody3D.

If left empty and node_a is set, the body is attached to a fixed StaticBody3D without collision shapes.

---

int **solver_priority** = `1`

- void **set_solver_priority**(value: int)
- int **get_solver_priority**()

The priority used to define which solver is executed first for multiple joints. The lower the value, the higher the priority.

Method Descriptions

RID **get_rid**() const

Returns the joint's internal RID from the PhysicsServer3D.

Label

Inherits: Control < CanvasItem < Node < Object

A control for displaying plain text.

Description

A control for displaying plain text. It gives you control over the horizontal and vertical alignment and can wrap the text inside the node's bounding rectangle. It doesn't support bold, italics, or other rich text formatting. For that, use RichTextLabel instead.

Tutorials

- 2D Dodge The Creeps Demo [https://godotengine.org/asset-library/asset/2712]

Properties

AutowrapMode	autowrap_mode	`0`
bool	clip_text	`false`
String	ellipsis_char	`"…"`
HorizontalAlignment	horizontal_alignment	`0`
BitField[JustificationFlag]	justification_flags	`163`
LabelSettings	label_settings	
String	language	`""`
int	lines_skipped	`0`

int	max_lines_visible	`-1`
MouseFilter	mouse_filter	`2` (overrides Control)
BitField[SizeFlags]	size_flags_vertical	`4` (overrides Control)
StructuredTextParser	structured_text_bidi_override	`0`
Array	structured_text_bidi_override_options	`[]`
PackedFloat32Array	tab_stops	`PackedFloat32Array()`
String	text	`""`
TextDirection	text_direction	`0`
OverrunBehavior	text_overrun_behavior	`0`
bool	uppercase	`false`
VerticalAlignment	vertical_alignment	`0`
int	visible_characters	`-1`
VisibleCharactersBehavior	visible_characters_behavior	`0`
float	visible_ratio	`1.0`

Methods

Rect2 get_character_bounds(pos: int) const

int	get_line_count() const
int	get_line_height(line: int = -1) const
int	get_total_character_count() const
int	get_visible_line_count() const

Theme Properties

Color	font_color	`Color(1, 1, 1, 1)`
Color	font_outline_color	`Color(0, 0, 0, 1)`
Color	font_shadow_color	`Color(0, 0, 0, 0)`
int	line_spacing	`3`
int	outline_size	`0`
int	shadow_offset_x	`1`
int	shadow_offset_y	`1`
int	shadow_outline_size	`1`
Font	font	
int	font_size	
StyleBox	normal	

Property Descriptions

AutowrapMode **autowrap_mode** = `0` 🔗

- void **set_autowrap_mode**(value: AutowrapMode)
- AutowrapMode **get_autowrap_mode**()

If set to something other than TextServer.AUTOWRAP_OFF, the text gets wrapped inside the node's bounding rectangle. If you resize the node, it will change its height automatically to show all the text. To see how each mode behaves, see AutowrapMode.

---

bool **clip_text** = `false` 🔗

- void **set_clip_text**(value: bool)
- bool **is_clipping_text**()

If `true`, the Label only shows the text that fits inside its bounding rectangle and will clip text horizontally.

String **ellipsis_char** = `"…"` 🔗

- void **set_ellipsis_char**(value: String)
- String **get_ellipsis_char**()

Ellipsis character used for text clipping.

HorizontalAlignment **horizontal_alignment** = `0` 🔗

- void **set_horizontal_alignment**(value: HorizontalAlignment)
- HorizontalAlignment **get_horizontal_alignment**()

Controls the text's horizontal alignment. Supports left, center, right, and fill, or justify. Set it to one of the HorizontalAlignment constants.

BitField[JustificationFlag] **justification_flags** = `163` 🔗

- void **set_justification_flags**(value: BitField[JustificationFlag])
- BitField[JustificationFlag] **get_justification_flags**()

Line fill alignment rules. For more info see JustificationFlag.

LabelSettings **label_settings** 🔗

- void **set_label_settings**(value: LabelSettings)
- LabelSettings **get_label_settings**()

A LabelSettings resource that can be shared between multiple **Label** nodes. Takes priority over theme properties.

String **language** = `""` 🔗

- void **set_language**(value: String)
- String **get_language**()

Language code used for line-breaking and text shaping algorithms, if left empty current locale is used instead.

---

int **lines_skipped** = `0` 🔗

- void **set_lines_skipped**(value: int)
- int **get_lines_skipped**()

The number of the lines ignored and not displayed from the start of the text value.

---

int **max_lines_visible** = `-1` 🔗

- void **set_max_lines_visible**(value: int)
- int **get_max_lines_visible**()

Limits the lines of text the node shows on screen.

StructuredTextParser **structured_text_bidi_override** = `0` 🔗

- void **set_structured_text_bidi_override**(value: StructuredTextParser)
- StructuredTextParser **get_structured_text_bidi_override**()

Set BiDi algorithm override for the structured text.

Array **structured_text_bidi_override_options** = `[]` 🔗

- void **set_structured_text_bidi_override_options**(value: Array)
- Array **get_structured_text_bidi_override_options**()

Set additional options for BiDi override.

PackedFloat32Array **tab_stops** = `PackedFloat32Array()` 🔗

- void **set_tab_stops**(value: PackedFloat32Array)
- PackedFloat32Array **get_tab_stops**()

Aligns text to the given tab-stops.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedFloat32Array for more details.

String **text** = `""` 🔗

- void **set_text**(value: String)
- String **get_text**()

The text to display on screen.

TextDirection **text_direction** = `0` 🔗

- void **set_text_direction**(value: TextDirection)
- TextDirection **get_text_direction**()

Base text writing direction.

OverrunBehavior **text_overrun_behavior** = `0` 🔗

- void **set_text_overrun_behavior**(value: OverrunBehavior)
- OverrunBehavior **get_text_overrun_behavior**()

Sets the clipping behavior when the text exceeds the node's bounding rectangle. See OverrunBehavior for a description of all modes.

---

bool **uppercase** = `false` 🔗

- void **set_uppercase**(value: bool)
- bool **is_uppercase**()

If `true`, all the text displays as UPPERCASE.

VerticalAlignment **vertical_alignment** = `0` 🔗

- void **set_vertical_alignment**(value: VerticalAlignment)
- VerticalAlignment **get_vertical_alignment**()

Controls the text's vertical alignment. Supports top, center, bottom, and fill. Set it to one of the VerticalAlignment constants.

---

int **visible_characters** = `-1` 🔗

- void **set_visible_characters**(value: int)
- int **get_visible_characters**()

The number of characters to display. If set to `-1`, all characters are displayed. This can be useful when animating the text appearing in a dialog box.

Note: Setting this property updates visible_ratio accordingly.

VisibleCharactersBehavior **visible_characters_behavior** = `0` 🔗

- void **set_visible_characters_behavior**(value: VisibleCharactersBehavior)
- VisibleCharactersBehavior **get_visible_characters_behavior**()

Sets the clipping behavior when visible_characters or visible_ratio is set. See VisibleCharactersBehavior for more info.

---

float **visible_ratio** = `1.0` 🔗

- void **set_visible_ratio**(value: float)
- float **get_visible_ratio**()

The fraction of characters to display, relative to the total number of characters (see get_total_character_count). If set to `1.0`, all characters are displayed. If set to `0.5`, only half of the characters will be displayed. This can be useful when animating the text appearing in a dialog box.

Note: Setting this property updates visible_characters accordingly.

Method Descriptions

Rect2 **get_character_bounds**(pos: int) const 🔗

Returns the bounding rectangle of the character at position `pos`. If the character is a non-visual character or `pos` is outside the valid range, an empty Rect2 is returned. If the character is a part of a composite grapheme, the bounding rectangle of the whole grapheme is returned.

int **get_line_count**() const

Returns the number of lines of text the Label has.

int **get_line_height**(line: int = -1) const

Returns the height of the line `line`.

If `line` is set to `-1`, returns the biggest line height.

If there are no lines, returns font size in pixels.

int **get_total_character_count**() const

Returns the total number of printable characters in the text (excluding spaces and newlines).

int **get_visible_line_count**() const

Returns the number of lines shown. Useful if the **Label**'s height cannot currently display all lines.

Theme Property Descriptions

Color **font_color** = `Color(1, 1, 1, 1)`

Default text Color of the **Label**.

Color **font_outline_color** = `Color(0, 0, 0, 1)`

The color of text outline.

Color **font_shadow_color** = `Color(0, 0, 0, 0)`

Color of the text's shadow effect.

int **line_spacing** = `3`

Vertical space between lines in multiline **Label**.

---

int **outline_size** = `0`

Text outline size.

Note: If using a font with FontFile.multichannel_signed_distance_field enabled, its FontFile.msdf_pixel_range must be set to at least *twice* the value of outline_size for outline rendering to look correct. Otherwise, the outline may appear to be cut off earlier than intended.

Note: Using a value that is larger than half the font size is not recommended, as the font outline may fail to be fully closed in this case.

---

int **shadow_offset_x** = `1`

The horizontal offset of the text's shadow.

---

int **shadow_offset_y** = `1`

The vertical offset of the text's shadow.

---

int **shadow_outline_size** = `1`

The size of the shadow outline.

Font **font**

Font used for the **Label**'s text.

---

int **font_size**

Font size of the **Label**'s text.

StyleBox **normal**

Background StyleBox for the **Label**.

Label3D

Inherits: GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A node for displaying plain text in 3D space.

Description

A node for displaying plain text in 3D space. By adjusting various properties of this node, you can configure things such as the text's appearance and whether it always faces the camera.

Tutorials

- 3D text

Properties

float	alpha_antialiasing_edge	`0.0`
AlphaAntiAliasing	alpha_antialiasing_mode	`0`
AlphaCutMode	alpha_cut	`0`
float	alpha_hash_scale	`1.0`
float	alpha_scissor_threshold	`0.5`
AutowrapMode	autowrap_mode	`0`
BillboardMode	billboard	`0`
ShadowCastingSetting	cast_shadow	`0` (overrides GeometryInstance3D)

bool	double_sided	`true`
bool	fixed_size	`false`
Font	font	
int	font_size	`32`
GIMode	gi_mode	`0` (overrides GeometryInstance3D)
HorizontalAlignment	horizontal_alignment	`1`
BitField[JustificationFlag]	justification_flags	`163`
String	language	`""`
float	line_spacing	`0.0`
Color	modulate	`Color(1, 1, 1, 1)`
bool	no_depth_test	`false`
Vector2	offset	`Vector2(0, 0)`
Color	outline_modulate	`Color(0, 0, 0, 1)`
int	outline_render_priority	`-1`
int	outline_size	`12`

float	pixel_size	`0.005`
int	render_priority	`0`
bool	shaded	`false`
StructuredTextParser	structured_text_bidi_override	`0`
Array	structured_text_bidi_override_options	`[]`
String	text	`""`
Direction	text_direction	`0`
TextureFilter	texture_filter	`3`
bool	uppercase	`false`
VerticalAlignment	vertical_alignment	`1`
float	width	`500.0`

Methods

TriangleMesh	generate_triangle_mesh() const
bool	get_draw_flag(flag: DrawFlags) const
void	set_draw_flag(flag: DrawFlags, enabled: bool)

Enumerations

enum **DrawFlags**: 🔗

DrawFlags **FLAG_SHADED** = `0`

If set, lights in the environment affect the label.

DrawFlags **FLAG_DOUBLE_SIDED** = `1`

If set, text can be seen from the back as well. If not, the text is invisible when looking at it from behind.

DrawFlags **FLAG_DISABLE_DEPTH_TEST** = `2`

Disables the depth test, so this object is drawn on top of all others. However, objects drawn after it in the draw order may cover it.

DrawFlags **FLAG_FIXED_SIZE** = `3`

Label is scaled by depth so that it always appears the same size on screen.

DrawFlags **FLAG_MAX** = `4`

Represents the size of the DrawFlags enum.

---

enum **AlphaCutMode**: 🔗

AlphaCutMode **ALPHA_CUT_DISABLED** = `0`

This mode performs standard alpha blending. It can display translucent areas, but transparency sorting issues may be visible when multiple transparent materials are overlapping. GeometryInstance3D.cast_shadow has no effect when this transparency mode is used; the **Label3D** will never cast shadows.

AlphaCutMode **ALPHA_CUT_DISCARD** = `1`

This mode only allows fully transparent or fully opaque pixels. Harsh edges will be visible unless some form of screen-space antialiasing is enabled (see ProjectSettings.rendering/anti_aliasing/quality/screen_space_aa). This mode is also known as *alpha testing* or *1-bit transparency*.

Note: This mode might have issues with anti-aliased fonts and outlines, try adjusting alpha_scissor_threshold or using MSDF font.

Note: When using text with overlapping glyphs (e.g., cursive scripts), this mode might have transparency sorting issues between the main text and the outline.

AlphaCutMode **ALPHA_CUT_OPAQUE_PREPASS** = `2`

This mode draws fully opaque pixels in the depth prepass. This is slower than ALPHA_CUT_DISABLED or ALPHA_CUT_DISCARD, but it allows displaying translucent areas and smooth edges while using proper sorting.

Note: When using text with overlapping glyphs (e.g., cursive scripts), this mode might have transparency sorting issues between the main text and the outline.

AlphaCutMode **ALPHA_CUT_HASH** = `3`

This mode draws cuts off all values below a spatially-deterministic threshold, the rest will remain opaque.

Property Descriptions

float **alpha_antialiasing_edge** = `0.0`

- void **set_alpha_antialiasing_edge**(value: float)
- float **get_alpha_antialiasing_edge**()

Threshold at which antialiasing will be applied on the alpha channel.

AlphaAntiAliasing **alpha_antialiasing_mode** = `0`

- void **set_alpha_antialiasing**(value: AlphaAntiAliasing)
- AlphaAntiAliasing **get_alpha_antialiasing**()

The type of alpha antialiasing to apply. See AlphaAntiAliasing.

AlphaCutMode **alpha_cut** = `0`

- void **set_alpha_cut_mode**(value: AlphaCutMode)
- AlphaCutMode **get_alpha_cut_mode**()

The alpha cutting mode to use for the sprite. See AlphaCutMode for possible values.

---

float **alpha_hash_scale** = `1.0`

- void **set_alpha_hash_scale**(value: float)

- float **get_alpha_hash_scale**()

The hashing scale for Alpha Hash. Recommended values between `0` and `2`.

---

float **alpha_scissor_threshold** = `0.5` 🔗

- void **set_alpha_scissor_threshold**(value: float)
- float **get_alpha_scissor_threshold**()

Threshold at which the alpha scissor will discard values.

AutowrapMode **autowrap_mode** = `0` 🔗

- void **set_autowrap_mode**(value: AutowrapMode)
- AutowrapMode **get_autowrap_mode**()

If set to something other than TextServer.AUTOWRAP_OFF, the text gets wrapped inside the node's bounding rectangle. If you resize the node, it will change its height automatically to show all the text. To see how each mode behaves, see AutowrapMode.

BillboardMode **billboard** = `0` 🔗

- void **set_billboard_mode**(value: BillboardMode)
- BillboardMode **get_billboard_mode**()

The billboard mode to use for the label. See BillboardMode for possible values.

---

bool **double_sided** = `true` 🔗

- void **set_draw_flag**(flag: DrawFlags, enabled: bool)
- bool **get_draw_flag**(flag: DrawFlags) const

If `true`, text can be seen from the back as well, if `false`, it is invisible when looking at it from behind.

---

bool **fixed_size** = `false` 🔗

- void **set_draw_flag**(flag: DrawFlags, enabled: bool)
- bool **get_draw_flag**(flag: DrawFlags) const

If `true`, the label is rendered at the same size regardless of distance.

Font **font** 🔗

- void **set_font**(value: Font)
- Font **get_font**()

Font configuration used to display text.

---

int **font_size** = `32` 🔗

- void **set_font_size**(value: int)
- int **get_font_size**()

Font size of the **Label3D**'s text. To make the font look more detailed when up close, increase font_size while decreasing pixel_size at the same time.

Higher font sizes require more time to render new characters, which can cause stuttering during gameplay.

HorizontalAlignment **horizontal_alignment** = `1` 🔗

- void **set_horizontal_alignment**(value: HorizontalAlignment)
- HorizontalAlignment **get_horizontal_alignment**()

Controls the text's horizontal alignment. Supports left, center, right, and fill, or justify. Set it to one of the HorizontalAlignment constants.

BitField[JustificationFlag] **justification_flags** = `163` 🔗

- void **set_justification_flags**(value: BitField[JustificationFlag])
- BitField[JustificationFlag] **get_justification_flags**()

Line fill alignment rules. For more info see JustificationFlag.

String **language** = `""` 🔗

- void **set_language**(value: String)
- String **get_language**()

Language code used for line-breaking and text shaping algorithms, if left empty current locale is used instead.

---

float **line_spacing** = `0.0` 🔗

- void **set_line_spacing**(value: float)
- float **get_line_spacing**()

Vertical space between lines in multiline **Label3D**.

Color **modulate** = `Color(1, 1, 1, 1)` 🔗

- void **set_modulate**(value: Color)
- Color **get_modulate**()

Text Color of the **Label3D**.

---

bool **no_depth_test** = `false` 🔗

- void **set_draw_flag**(flag: DrawFlags, enabled: bool)
- bool **get_draw_flag**(flag: DrawFlags) const

If `true`, depth testing is disabled and the object will be drawn in render order.

Vector2 **offset** = `Vector2(0, 0)` 🔗

- void **set_offset**(value: Vector2)
- Vector2 **get_offset**()

The text drawing offset (in pixels).

Color **outline_modulate** = `Color(0, 0, 0, 1)` 🔗

- void **set_outline_modulate**(value: Color)
- Color **get_outline_modulate**()

The tint of text outline.

---

int **outline_render_priority** = `-1` 🔗

- void **set_outline_render_priority**(value: int)
- int **get_outline_render_priority**()

Sets the render priority for the text outline. Higher priority objects will be sorted in front of lower priority objects.

Note: This only applies if alpha_cut is set to ALPHA_CUT_DISABLED (default value).

Note: This only applies to sorting of transparent objects. This will not impact how transparent objects are sorted relative to opaque objects. This is because opaque objects are not sorted, while transparent objects are sorted from back to front (subject to priority).

---

int **outline_size** = `12`

- void **set_outline_size**(value: int)
- int **get_outline_size**()

Text outline size.

---

float **pixel_size** = `0.005`

- void **set_pixel_size**(value: float)
- float **get_pixel_size**()

The size of one pixel's width on the label to scale it in 3D. To make the font look more detailed when up close, increase font_size while decreasing pixel_size at the same time.

---

int **render_priority** = `0`

- void **set_render_priority**(value: int)
- int **get_render_priority**()

Sets the render priority for the text. Higher priority objects will be sorted in front of lower priority objects.

Note: This only applies if alpha_cut is set to ALPHA_CUT_DISABLED (default value).

Note: This only applies to sorting of transparent objects. This will not impact how transparent objects are sorted relative to opaque objects. This is because opaque objects are not sorted, while transparent objects are sorted from back to front (subject to priority).

---

bool **shaded** = `false`

- void **set_draw_flag**(flag: DrawFlags, enabled: bool)
- bool **get_draw_flag**(flag: DrawFlags) const

If `true`, the Light3D in the Environment has effects on the label.

StructuredTextParser **structured_text_bidi_override** = `0`

- void **set_structured_text_bidi_override**(value: StructuredTextParser)
- StructuredTextParser **get_structured_text_bidi_override**()

Set BiDi algorithm override for the structured text.

Array **structured_text_bidi_override_options** = `[]`

- void **set_structured_text_bidi_override_options**(value: Array)
- Array **get_structured_text_bidi_override_options**()

Set additional options for BiDi override.

String **text** = `""`

- void **set_text**(value: String)
- String **get_text**()

The text to display on screen.

Direction **text_direction** = `0`

- void **set_text_direction**(value: Direction)
- Direction **get_text_direction**()

Base text writing direction.

TextureFilter **texture_filter** = `3`

- void **set_texture_filter**(value: TextureFilter)
- TextureFilter **get_texture_filter**()

Filter flags for the texture. See TextureFilter for options.

bool **uppercase** = `false`

- void **set_uppercase**(value: bool)
- bool **is_uppercase**()

If `true`, all the text displays as UPPERCASE.

VerticalAlignment **vertical_alignment** = `1`

- void **set_vertical_alignment**(value: VerticalAlignment)
- VerticalAlignment **get_vertical_alignment**()

Controls the text's vertical alignment. Supports top, center, bottom. Set it to one of the VerticalAlignment constants.

---

float **width** = `500.0`

- void **set_width**(value: float)
- float **get_width**()

Text width (in pixels), used for autowrap and fill alignment.

Method Descriptions

TriangleMesh **generate_triangle_mesh**() const

Returns a TriangleMesh with the label's vertices following its current configuration (such as its pixel_size).

---

bool **get_draw_flag**(flag: DrawFlags) const

Returns the value of the specified flag.

---

void **set_draw_flag**(flag: DrawFlags, enabled: bool)

If `true`, the specified flag will be enabled. See DrawFlags for a list of flags.

Light2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Inherited By: DirectionalLight2D, PointLight2D

Casts light in a 2D environment.

Description

Casts light in a 2D environment. A light is defined as a color, an energy value, a mode (see constants), and various other parameters (range and shadows-related).

Tutorials

- 2D lights and shadows

Properties

BlendMode	blend_mode	`0`
Color	color	`Color(1, 1, 1, 1)`
bool	editor_only	`false`
bool	enabled	`true`

float	energy	`1.0`
int	range_item_cull_mask	`1`
int	range_layer_max	`0`
int	range_layer_min	`0`
int	range_z_max	`1024`
int	range_z_min	`-1024`
Color	shadow_color	`Color(0, 0, 0, 0)`
bool	shadow_enabled	`false`
ShadowFilter	shadow_filter	`0`
float	shadow_filter_smooth	`0.0`
int	shadow_item_cull_mask	`1`

Methods

float get_height() const

---

void set_height(height: float)

Enumerations

enum **ShadowFilter**: 🔗

ShadowFilter **SHADOW_FILTER_NONE** = `0`

No filter applies to the shadow map. This provides hard shadow edges and is the fastest to render. See shadow_filter.

ShadowFilter **SHADOW_FILTER_PCF5** = `1`

Percentage closer filtering (5 samples) applies to the shadow map. This is slower compared to hard shadow rendering. See shadow_filter.

ShadowFilter **SHADOW_FILTER_PCF13** = `2`

Percentage closer filtering (13 samples) applies to the shadow map. This is the slowest shadow filtering mode, and should be used sparingly. See shadow_filter.

---

enum **BlendMode**: 🔗

BlendMode **BLEND_MODE_ADD** = `0`

Adds the value of pixels corresponding to the Light2D to the values of pixels under it. This is the common behavior of a light.

BlendMode **BLEND_MODE_SUB** = `1`

Subtracts the value of pixels corresponding to the Light2D to the values of pixels under it, resulting in inversed light effect.

BlendMode **BLEND_MODE_MIX** = `2`

Mix the value of pixels corresponding to the Light2D to the values of pixels under it by linear interpolation.

Property Descriptions

BlendMode **blend_mode** = `0`

- void **set_blend_mode**(value: BlendMode)
- BlendMode **get_blend_mode**()

The Light2D's blend mode. See BlendMode constants for values.

Color **color** = `Color(1, 1, 1, 1)`

- void **set_color**(value: Color)
- Color **get_color**()

The Light2D's Color.

bool **editor_only** = `false`

- void **set_editor_only**(value: bool)
- bool **is_editor_only**()

If `true`, Light2D will only appear when editing the scene.

bool **enabled** = `true` 🔗

- void **set_enabled**(value: bool)
- bool **is_enabled**()

If `true`, Light2D will emit light.

---

float **energy** = `1.0` 🔗

- void **set_energy**(value: float)
- float **get_energy**()

The Light2D's energy value. The larger the value, the stronger the light.

---

int **range_item_cull_mask** = `1` 🔗

- void **set_item_cull_mask**(value: int)
- int **get_item_cull_mask**()

The layer mask. Only objects with a matching CanvasItem.light_mask will be affected by the Light2D. See also shadow_item_cull_mask, which affects which objects can cast shadows.

Note: range_item_cull_mask is ignored by DirectionalLight2D, which will always light a 2D node regardless of the 2D node's CanvasItem.light_mask.

---

int **range_layer_max** = `0` 🔗

- void **set_layer_range_max**(value: int)
- int **get_layer_range_max**()

Maximum layer value of objects that are affected by the Light2D.

---

int **range_layer_min** = `0` 🔗

- void **set_layer_range_min**(value: int)
- int **get_layer_range_min**()

Minimum layer value of objects that are affected by the Light2D.

---

int **range_z_max** = `1024` 🔗

- void **set_z_range_max**(value: int)
- int **get_z_range_max**()

Maximum `z` value of objects that are affected by the Light2D.

---

int **range_z_min** = `-1024` 🔗

- void **set_z_range_min**(value: int)
- int **get_z_range_min**()

Minimum `z` value of objects that are affected by the Light2D.

Color **shadow_color** = `Color(0, 0, 0, 0)` 🔗

- void **set_shadow_color**(value: Color)
- Color **get_shadow_color**()

Color of shadows cast by the Light2D.

---

bool **shadow_enabled** = `false` 🔗

- void **set_shadow_enabled**(value: bool)
- bool **is_shadow_enabled**()

If `true`, the Light2D will cast shadows.

ShadowFilter **shadow_filter** = `0` 🔗

- void **set_shadow_filter**(value: ShadowFilter)
- ShadowFilter **get_shadow_filter**()

Shadow filter type. See ShadowFilter for possible values.

---

float **shadow_filter_smooth** = `0.0` 🔗

- void **set_shadow_smooth**(value: float)
- float **get_shadow_smooth**()

Smoothing value for shadows. Higher values will result in softer shadows, at the cost of visible streaks that can appear in shadow rendering. shadow_filter_smooth only has an effect if shadow_filter is SHADOW_FILTER_PCF5 or SHADOW_FILTER_PCF13.

---

int **shadow_item_cull_mask** = `1` 🔗

- void **set_item_shadow_cull_mask**(value: int)
- int **get_item_shadow_cull_mask**()

The shadow mask. Used with LightOccluder2D to cast shadows. Only occluders with a matching CanvasItem.light_mask will cast shadows. See also range_item_cull_mask, which affects which objects can *receive* the light.

Method Descriptions

float **get_height**() const

Returns the light's height, which is used in 2D normal mapping. See PointLight2D.height and DirectionalLight2D.height.

---

void **set_height**(height: float)

Sets the light's height, which is used in 2D normal mapping. See PointLight2D.height and DirectionalLight2D.height.

Light3D

Inherits: VisualInstance3D **<** Node3D **<** Node **<** Object

Inherited By: DirectionalLight3D, OmniLight3D, SpotLight3D

Provides a base class for different kinds of light nodes.

Description

Light3D is the *abstract* base class for light nodes. As it can't be instantiated, it shouldn't be used directly. Other types of light nodes inherit from it. Light3D contains the common variables and parameters used for lighting.

Tutorials

- 3D lights and shadows
- Faking global illumination
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

float	distance_fade_begin	`40.0`
bool	distance_fade_enabled	`false`

float	distance_fade_length	`10.0`
float	distance_fade_shadow	`50.0`
bool	editor_only	`false`
float	light_angular_distance	`0.0`
BakeMode	light_bake_mode	`2`
Color	light_color	`Color(1, 1, 1, 1)`
int	light_cull_mask	`4294967295`
float	light_energy	`1.0`
float	light_indirect_energy	`1.0`
float	light_intensity_lumens	
float	light_intensity_lux	
bool	light_negative	`false`
Texture2D	light_projector	

float	light_size	`0.0`
float	light_specular	`0.5`
float	light_temperature	
float	light_volumetric_fog_energy	`1.0`
float	shadow_bias	`0.1`
float	shadow_blur	`1.0`
bool	shadow_enabled	`false`
float	shadow_normal_bias	`2.0`
float	shadow_opacity	`1.0`
bool	shadow_reverse_cull_face	`false`
float	shadow_transmittance_bias	`0.05`

Methods

Color	get_correlated_color() const
float	get_param(param: Param) const
void	set_param(param: Param, value: float)

Enumerations

enum **Param**: 🔗

Param **PARAM_ENERGY** = `0`

Constant for accessing light_energy.

Param **PARAM_INDIRECT_ENERGY** = `1`

Constant for accessing light_indirect_energy.

Param **PARAM_VOLUMETRIC_FOG_ENERGY** = `2`

Constant for accessing light_volumetric_fog_energy.

Param **PARAM_SPECULAR** = `3`

Constant for accessing light_specular.

Param **PARAM_RANGE** = `4`

Constant for accessing OmniLight3D.omni_range or SpotLight3D.spot_range.

Param **PARAM_SIZE** = `5`

Constant for accessing light_size.

Param **PARAM_ATTENUATION** = `6`

Constant for accessing OmniLight3D.omni_attenuation or SpotLight3D.spot_attenuation.

Param **PARAM_SPOT_ANGLE** = `7`

Constant for accessing SpotLight3D.spot_angle.

Param **PARAM_SPOT_ATTENUATION** = `8`

Constant for accessing SpotLight3D.spot_angle_attenuation.

Param **PARAM_SHADOW_MAX_DISTANCE** = `9`

Constant for accessing DirectionalLight3D.directional_shadow_max_distance.

Param **PARAM_SHADOW_SPLIT_1_OFFSET** = `10`

Constant for accessing DirectionalLight3D.directional_shadow_split_1.

Param **PARAM_SHADOW_SPLIT_2_OFFSET** = `11`

Constant for accessing DirectionalLight3D.directional_shadow_split_2.

Param **PARAM_SHADOW_SPLIT_3_OFFSET** = `12`

Constant for accessing DirectionalLight3D.directional_shadow_split_3.

Param **PARAM_SHADOW_FADE_START** = `13`

Constant for accessing DirectionalLight3D.directional_shadow_fade_start.

Param **PARAM_SHADOW_NORMAL_BIAS** = `14`

Constant for accessing shadow_normal_bias.

Param **PARAM_SHADOW_BIAS** = `15`

Constant for accessing shadow_bias.

Param **PARAM_SHADOW_PANCAKE_SIZE** = `16`

Constant for accessing DirectionalLight3D.directional_shadow_pancake_size.

Param **PARAM_SHADOW_OPACITY** = `17`

Constant for accessing shadow_opacity.

Param **PARAM_SHADOW_BLUR** = `18`

Constant for accessing shadow_blur.

Param **PARAM_TRANSMITTANCE_BIAS** = `19`

Constant for accessing shadow_transmittance_bias.

Param **PARAM_INTENSITY** = `20`

Constant for accessing light_intensity_lumens and light_intensity_lux. Only used when ProjectSettings.rendering/lights_and_shadows/use_physical_light_units is `true`.

Param **PARAM_MAX** = `21`

Represents the size of the Param enum.

---

enum **BakeMode**: 🔗

BakeMode **BAKE_DISABLED** = `0`

Light is ignored when baking. This is the fastest mode, but the light will be taken into account when baking global illumination. This mode should generally be used for dynamic lights that change quickly, as the effect of global illumination is less noticeable on those lights.

Note: Hiding a light does *not* affect baking LightmapGI. Hiding a light will still affect baking VoxelGI and SDFGI (see Environment.sdfgi_enabled).

BakeMode **BAKE_STATIC** = `1`

Light is taken into account in static baking (VoxelGI, LightmapGI, SDFGI (Environment.sdfgi_enabled)). The light can be moved around or modified, but its global illumination will not update in real-time. This is suitable for subtle changes (such as flickering torches), but generally not large changes such as toggling a light on and off.

Note: The light is not baked in LightmapGI if editor_only is `true`.

BakeMode **BAKE_DYNAMIC** = `2`

Light is taken into account in dynamic baking (VoxelGI and SDFGI (Environment.sdfgi_enabled) only). The light can be moved around or modified with global illumination updating in real-time. The light's global illumination appearance will be slightly different compared to BAKE_STATIC. This has a greater performance cost compared to BAKE_STATIC. When using SDFGI, the update speed of dynamic lights is affected by ProjectSettings.rendering/global_illumination/sdfgi/frames_to_update_lights.

Property Descriptions

float **distance_fade_begin** = `40.0` 🔗

- void **set_distance_fade_begin**(value: float)
- float **get_distance_fade_begin**()

The distance from the camera at which the light begins to fade away (in 3D units).

Note: Only effective for OmniLight3D and SpotLight3D.

---

bool **distance_fade_enabled** = `false` 🔗

- void **set_enable_distance_fade**(value: bool)
- bool **is_distance_fade_enabled**()

If `true`, the light will smoothly fade away when far from the active Camera3D starting at distance_fade_begin. This acts as a form of level of detail (LOD). The light will fade out over distance_fade_begin + distance_fade_length, after which it will be culled and not sent to the shader at all. Use this to reduce the number of active lights in a scene and thus improve performance.

Note: Only effective for OmniLight3D and SpotLight3D.

---

float **distance_fade_length** = `10.0` 🔗

- void **set_distance_fade_length**(value: float)
- float **get_distance_fade_length**()

Distance over which the light and its shadow fades. The light's energy and shadow's opacity is progressively reduced over this distance and is completely invisible at the end.

Note: Only effective for OmniLight3D and SpotLight3D.

---

float **distance_fade_shadow** = `50.0` 🔗

- void **set_distance_fade_shadow**(value: float)
- float **get_distance_fade_shadow**()

The distance from the camera at which the light's shadow cuts off (in 3D units). Set this to a value lower than distance_fade_begin + distance_fade_length to further improve performance, as shadow rendering is often more expensive than light rendering itself.

Note: Only effective for OmniLight3D and SpotLight3D, and only when shadow_enabled is `true`.

---

bool **editor_only** = `false` 🔗

- void **set_editor_only**(value: bool)
- bool **is_editor_only**()

If `true`, the light only appears in the editor and will not be visible at runtime. If `true`, the light will never be baked in LightmapGI regardless of its light_bake_mode.

---

float **light_angular_distance** = `0.0` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The light's angular size in degrees. Increasing this will make shadows softer at greater distances (also called percentage-closer soft shadows, or PCSS). Only available for DirectionalLight3Ds. For reference, the Sun from the Earth is approximately `0.5`. Increasing this value above `0.0` for lights with shadows enabled will have a noticeable performance cost due to PCSS.

Note: light_angular_distance is not affected by Node3D.scale (the light's scale or its parent's scale).

Note: PCSS for directional lights is only supported in the Forward+ rendering method, not Mobile or Compatibility.

BakeMode **light_bake_mode** = `2` 🔗

- void **set_bake_mode**(value: BakeMode)
- BakeMode **get_bake_mode**()

The light's bake mode. This will affect the global illumination techniques that have an effect on the light's rendering. See BakeMode.

Note: Meshes' global illumination mode will also affect the global illumination rendering. See GeometryInstance3D.gi_mode.

Color **light_color** = `Color(1, 1, 1, 1)` 🔗

- void **set_color**(value: Color)
- Color **get_color**()

The light's color. An *overbright* color can be used to achieve a result equivalent to increasing the light's light_energy.

int **light_cull_mask** = `4294967295` 🔗

- void **set_cull_mask**(value: int)
- int **get_cull_mask**()

The light will affect objects in the selected layers.

float **light_energy** = `1.0` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The light's strength multiplier (this is not a physical unit). For OmniLight3D and SpotLight3D, changing this value will only change the light color's intensity, not the light's radius.

float **light_indirect_energy** = `1.0` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

Secondary multiplier used with indirect light (light bounces). Used with VoxelGI and SDFGI (see Environment.sdfgi_enabled).

Note: This property is ignored if light_energy is equal to `0.0`, as the light won't be present at all in the GI shader.

float **light_intensity_lumens** 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

Used by positional lights (OmniLight3D and SpotLight3D) when ProjectSettings.rendering/lights_and_shadows/use_physical_light_units is `true`. Sets the intensity of the light source measured in Lumens. Lumens are a measure of luminous flux, which is the total amount of visible light emitted by a light source per unit of time.

For SpotLight3Ds, we assume that the area outside the visible cone is surrounded by a perfect light absorbing material. Accordingly, the apparent brightness of the cone area does not change as the cone increases and decreases in size.

A typical household lightbulb can range from around 600 lumens to 1,200 lumens, a candle is about 13 lumens, while a streetlight can be approximately 60,000 lumens.

---

float **light_intensity_lux** 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

Used by DirectionalLight3Ds when ProjectSettings.rendering/lights_and_shadows/use_physical_light_units is `true`. Sets the intensity of the light source measured in Lux. Lux is a measure of luminous flux per unit area, it is equal to one lumen per square meter. Lux is the measure of how much light hits a surface at a given time.

On a clear sunny day a surface in direct sunlight may be approximately 100,000 lux, a typical room in a home may be approximately 50 lux, while the moonlit ground may be approximately 0.1 lux.

---

bool **light_negative** = `false` 🔗

- void **set_negative**(value: bool)
- bool **is_negative**()

If `true`, the light's effect is reversed, darkening areas and casting bright shadows.

Texture2D **light_projector**

- void **set_projector**(value: Texture2D)
- Texture2D **get_projector**()

Texture2D projected by light. shadow_enabled must be on for the projector to work. Light projectors make the light appear as if it is shining through a colored but transparent object, almost like light shining through stained-glass.

Note: Unlike BaseMaterial3D whose filter mode can be adjusted on a per-material basis, the filter mode for light projector textures is set globally with ProjectSettings.rendering/textures/light_projectors/filter.

Note: Light projector textures are only supported in the Forward+ and Mobile rendering methods, not Compatibility.

---

float **light_size** = `0.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The size of the light in Godot units. Only available for OmniLight3Ds and SpotLight3Ds. Increasing this value will make the light fade out slower and shadows appear blurrier (also called percentage-closer soft shadows, or PCSS). This can be used to simulate area lights to an extent. Increasing this value above `0.0` for lights with shadows enabled will have a noticeable performance cost due to PCSS.

Note: light_size is not affected by Node3D.scale (the light's scale or its parent's scale).

Note: PCSS for positional lights is only supported in the Forward+ and Mobile rendering methods, not Compatibility.

---

float **light_specular** = `0.5` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The intensity of the specular blob in objects affected by the light. At `0`, the light becomes a pure diffuse light. When not baking emission, this can be used to avoid unrealistic reflections when placing lights above an emissive surface.

---

float **light_temperature** 🔗

- void **set_temperature**(value: float)
- float **get_temperature**()

Sets the color temperature of the light source, measured in Kelvin. This is used to calculate a correlated color temperature which tints the light_color.

The sun on a cloudy day is approximately 6500 Kelvin, on a clear day it is between 5500 to 6000 Kelvin, and on a clear day at sunrise or sunset it ranges to around 1850 Kelvin.

---

float **light_volumetric_fog_energy** = `1.0` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

Secondary multiplier multiplied with light_energy then used with the Environment's volumetric fog (if enabled). If set to `0.0`, computing volumetric fog will be skipped for this light, which can improve performance for large amounts of lights when volumetric fog is enabled.

Note: To prevent short-lived dynamic light effects from poorly interacting with volumetric fog, lights used in those effects should have light_volumetric_fog_energy set to `0.0` unless Environment.volumetric_fog_temporal_reprojection_enabled is disabled (or unless the reprojection amount is significantly lowered).

---

float **shadow_bias** = `0.1`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

Used to adjust shadow appearance. Too small a value results in self-shadowing ("shadow acne"), while too large a value causes shadows to separate from casters ("peter-panning"). Adjust as needed.

---

float **shadow_blur** = `1.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

Blurs the edges of the shadow. Can be used to hide pixel artifacts in low-resolution shadow maps. A high value can impact performance, make shadows appear grainy and can cause other unwanted artifacts. Try to keep as near default as possible.

---

bool **shadow_enabled** = `false`

- void **set_shadow**(value: bool)
- bool **has_shadow**()

If `true`, the light will cast real-time shadows. This has a significant performance cost. Only enable shadow rendering when it makes a noticeable difference in the scene's appearance, and consider using distance_fade_enabled to hide the light when far away from the Camera3D.

---

float **shadow_normal_bias** = `2.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

Offsets the lookup into the shadow map by the object's normal. This can be used to reduce self-shadowing artifacts without using shadow_bias. In practice, this value should be tweaked along with shadow_bias to reduce artifacts as much as possible.

---

float **shadow_opacity** = `1.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The opacity to use when rendering the light's shadow map. Values lower than `1.0` make the light appear through shadows. This can be used to fake global illumination at a low performance cost.

---

bool **shadow_reverse_cull_face** = `false`

- void **set_shadow_reverse_cull_face**(value: bool)
- bool **get_shadow_reverse_cull_face**()

If `true`, reverses the backface culling of the mesh. This can be useful when you have a flat mesh that has a light behind it. If you need to cast a shadow on both sides of the mesh, set the mesh to use double-sided shadows with GeometryInstance3D.SHADOW_CASTING_SETTING_DOUBLE_SIDED.

---

float **shadow_transmittance_bias** = `0.05`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

There is currently no description for this property. Please help us by contributing one!

Method Descriptions

Color **get_correlated_color**() const

Returns the Color of an idealized blackbody at the given light_temperature. This value is calculated internally based on the light_temperature. This Color is multiplied by light_color before being sent to the RenderingServer.

---

float **get_param**(param: Param) const

Returns the value of the specified Param parameter.

---

void **set_param**(param: Param, value: float)

Sets the value of the specified Param parameter.

LightmapGI

Inherits: VisualInstance3D **<** Node3D **<** Node **<** Object

Computes and stores baked lightmaps for fast global illumination.

Description

The **LightmapGI** node is used to compute and store baked lightmaps. Lightmaps are used to provide high-quality indirect lighting with very little light leaking. **LightmapGI** can also provide rough reflections using spherical harmonics if directional is enabled. Dynamic objects can receive indirect lighting thanks to *light probes*, which can be automatically placed by setting generate_probes_subdiv to a value other than GENERATE_PROBES_DISABLED. Additional lightmap probes can also be added by creating LightmapProbe nodes. The downside is that lightmaps are fully static and cannot be baked in an exported project. Baking a **LightmapGI** node is also slower compared to VoxelGI.

Procedural generation: Lightmap baking functionality is only available in the editor. This means **LightmapGI** is not suited to procedurally generated or user-built levels. For procedurally generated or user-built levels, use VoxelGI or SDFGI instead (see Environment.sdfgi_enabled).

Performance: LightmapGI provides the best possible run-time performance for global illumination. It is suitable for low-end hardware including integrated graphics and mobile devices.

Note: Due to how lightmaps work, most properties only have a visible effect once lightmaps are baked again.

Note: Lightmap baking on CSGShape3Ds and PrimitiveMeshes is not supported, as these cannot store UV2 data required for baking.

Note: If no custom lightmappers are installed, **LightmapGI** can only be baked from devices that support the Forward+ or Mobile rendering backends.

Tutorials

- Using Lightmap global illumination

Properties

float	bias	`0.0005`
float	bounce_indirect_energy	`1.0`
int	bounces	`3`
CameraAttributes	camera_attributes	
int	denoiser_range	`10`
float	denoiser_strength	`0.1`
bool	directional	`false`

Color	environment_custom_color	
float	environment_custom_energy	
Sky	environment_custom_sky	
EnvironmentMode	environment_mode	`1`
GenerateProbes	generate_probes_subdiv	`2`
bool	interior	`false`
LightmapGIData	light_data	
int	max_texture_size	`16384`
BakeQuality	quality	`1`
float	texel_scale	`1.0`
bool	use_denoiser	`true`
bool	use_texture_for_bounces	`true`

Enumerations

enum **BakeQuality**: 🔗

BakeQuality **BAKE_QUALITY_LOW** = `0`

Low bake quality (fastest bake times). The quality of this preset can be adjusted by changing ProjectSettings.rendering/lightmapping/bake_quality/low_quality_ray_count and ProjectSettings.rendering/lightmapping/bake_quality/low_quality_probe_ray_count.

BakeQuality **BAKE_QUALITY_MEDIUM** = `1`

Medium bake quality (fast bake times). The quality of this preset can be adjusted by changing ProjectSettings.rendering/lightmapping/bake_quality/medium_quality_ray_count and ProjectSettings.rendering/lightmapping/bake_quality/medium_quality_probe_ray_count.

BakeQuality **BAKE_QUALITY_HIGH** = `2`

High bake quality (slow bake times). The quality of this preset can be adjusted by changing ProjectSettings.rendering/lightmapping/bake_quality/high_quality_ray_count and ProjectSettings.rendering/lightmapping/bake_quality/high_quality_probe_ray_count.

BakeQuality **BAKE_QUALITY_ULTRA** = `3`

Highest bake quality (slowest bake times). The quality of this preset can be adjusted by changing ProjectSettings.rendering/lightmapping/bake_quality/ultra_quality_ra y_count and ProjectSettings.rendering/lightmapping/bake_quality/ultra_quality_probe_ray_count.

enum **GenerateProbes**: 🔗

GenerateProbes **GENERATE_PROBES_DISABLED** = `0`

Don't generate lightmap probes for lighting dynamic objects.

GenerateProbes **GENERATE_PROBES_SUBDIV_4** = `1`

Lowest level of subdivision (fastest bake times, smallest file sizes).

GenerateProbes **GENERATE_PROBES_SUBDIV_8** = `2`

Low level of subdivision (fast bake times, small file sizes).

GenerateProbes **GENERATE_PROBES_SUBDIV_16** = `3`

High level of subdivision (slow bake times, large file sizes).

GenerateProbes **GENERATE_PROBES_SUBDIV_32** = `4`

Highest level of subdivision (slowest bake times, largest file sizes).

enum **BakeError**: 🔗

BakeError **BAKE_ERROR_OK** = `0`

Lightmap baking was successful.

BakeError **BAKE_ERROR_NO_SCENE_ROOT** = `1`

Lightmap baking failed because the root node for the edited scene could not be accessed.

BakeError **BAKE_ERROR_FOREIGN_DATA** = `2`

Lightmap baking failed as the lightmap data resource is embedded in a foreign resource.

BakeError **BAKE_ERROR_NO_LIGHTMAPPER** = `3`

Lightmap baking failed as there is no lightmapper available in this Godot build.

BakeError **BAKE_ERROR_NO_SAVE_PATH** = `4`

Lightmap baking failed as the LightmapGIData save path isn't configured in the resource.

BakeError **BAKE_ERROR_NO_MESHES** = `5`

Lightmap baking failed as there are no meshes whose GeometryInstance3D.gi_mode is GeometryInstance3D.GI_MODE_STATIC and with valid UV2 mapping in the current scene. You may need to select 3D scenes in the Import dock and change their global illumination mode accordingly.

BakeError **BAKE_ERROR_MESHES_INVALID** = `6`

Lightmap baking failed as the lightmapper failed to analyze some of the meshes marked as static for baking.

BakeError **BAKE_ERROR_CANT_CREATE_IMAGE** = `7`

Lightmap baking failed as the resulting image couldn't be saved or imported by Godot after it was saved.

BakeError **BAKE_ERROR_USER_ABORTED** = `8`

The user aborted the lightmap baking operation (typically by clicking the **Cancel** button in the progress dialog).

BakeError **BAKE_ERROR_TEXTURE_SIZE_TOO_SMALL** = `9`

Lightmap baking failed as the maximum texture size is too small to fit some of the meshes marked for baking.

BakeError **BAKE_ERROR_LIGHTMAP_TOO_SMALL** = `10`

Lightmap baking failed as the lightmap is too small.

BakeError **BAKE_ERROR_ATLAS_TOO_SMALL** = `11`

Lightmap baking failed as the lightmap was unable to fit into an atlas.

---

enum **EnvironmentMode**: 🔗

EnvironmentMode **ENVIRONMENT_MODE_DISABLED** = `0`

Ignore environment lighting when baking lightmaps.

EnvironmentMode **ENVIRONMENT_MODE_SCENE** = `1`

Use the scene's environment lighting when baking lightmaps.

Note: If baking lightmaps in a scene with no WorldEnvironment node, this will act like ENVIRONMENT_MODE_DISABLED. The editor's preview sky and sun is *not* taken into account by **LightmapGI** when baking lightmaps.

EnvironmentMode **ENVIRONMENT_MODE_CUSTOM_SKY** = `2`

Use environment_custom_sky as a source of environment lighting when baking lightmaps.

EnvironmentMode **ENVIRONMENT_MODE_CUSTOM_COLOR** = `3`

Use environment_custom_color multiplied by environment_custom_energy as a constant source of environment lighting when baking lightmaps.

Property Descriptions

float **bias** = `0.0005`

- void **set_bias**(value: float)
- float **get_bias**()

The bias to use when computing shadows. Increasing bias can fix shadow acne on the resulting baked lightmap, but can introduce peter-panning (shadows not connecting to their casters). Real-time Light3D shadows are not affected by this bias property.

---

float **bounce_indirect_energy** = `1.0`

- void **set_bounce_indirect_energy**(value: float)
- float **get_bounce_indirect_energy**()

The energy multiplier for each bounce. Higher values will make indirect lighting brighter. A value of `1.0` represents physically accurate behavior, but higher values can be used to make indirect lighting propagate more visibly when using a low number of bounces. This can be used to speed up bake times by lowering the number of bounces then increasing bounce_indirect_energy.

Note: bounce_indirect_energy only has an effect if bounces is set to a value greater than or equal to `1`.

---

int **bounces** = `3`

- void **set_bounces**(value: int)
- int **get_bounces**()

Number of light bounces that are taken into account during baking. Higher values result in brighter, more realistic lighting, at the cost of longer bake times. If set to `0`, only environment lighting, direct light and emissive lighting is baked.

CameraAttributes **camera_attributes** 🔗

- void **set_camera_attributes**(value: CameraAttributes)
- CameraAttributes **get_camera_attributes**()

The CameraAttributes resource that specifies exposure levels to bake at. Auto-exposure and non exposure properties will be ignored. Exposure settings should be used to reduce the dynamic range present when baking. If exposure is too high, the **LightmapGI** will have banding artifacts or may have over-exposure artifacts.

---

int **denoiser_range** = `10` 🔗

- void **set_denoiser_range**(value: int)
- int **get_denoiser_range**()

The distance in pixels from which the denoiser samples. Lower values preserve more details, but may give blotchy results if the lightmap quality is not high enough. Only effective if use_denoiser is `true` and ProjectSettings.rendering/lightmapping/denoising/denoiser is set to JNLM.

---

float **denoiser_strength** = `0.1` 🔗

- void **set_denoiser_strength**(value: float)
- float **get_denoiser_strength**()

The strength of denoising step applied to the generated lightmaps. Only effective if use_denoiser is `true` and ProjectSettings.rendering/lightmapping/denoising/denoiser is set to JNLM.

---

bool **directional** = `false` 🔗

- void **set_directional**(value: bool)
- bool **is_directional**()

If `true`, bakes lightmaps to contain directional information as spherical harmonics. This results in more realistic lighting appearance, especially with normal mapped materials and for lights that have their direct light baked (Light3D.light_bake_mode set to Light3D.BAKE_STATIC and with Light3D.editor_only set to `false`). The directional information is also used to provide rough reflections for static and dynamic objects. This has a small run-time performance cost as the shader has to perform more work to interpret the direction information from the lightmap. Directional lightmaps also take longer to bake and result in larger file sizes.

Note: The property's name has no relationship with DirectionalLight3D. directional works with all light types.

Color **environment_custom_color** 🔗

- void **set_environment_custom_color**(value: Color)
- Color **get_environment_custom_color**()

The color to use for environment lighting. Only effective if environment_mode is ENVIRONMENT_MODE_CUSTOM_COLOR.

---

float **environment_custom_energy** 🔗

- void **set_environment_custom_energy**(value: float)
- float **get_environment_custom_energy**()

The color multiplier to use for environment lighting. Only effective if environment_mode is ENVIRONMENT_MODE_CUSTOM_COLOR.

Sky **environment_custom_sky** 🔗

- void **set_environment_custom_sky**(value: Sky)
- Sky **get_environment_custom_sky**()

The sky to use as a source of environment lighting. Only effective if environment_mode is ENVIRONMENT_MODE_CUSTOM_SKY.

EnvironmentMode **environment_mode** = `1` 🔗

- void **set_environment_mode**(value: EnvironmentMode)
- EnvironmentMode **get_environment_mode**()

The environment mode to use when baking lightmaps.

GenerateProbes **generate_probes_subdiv** = `2` 🔗

- void **set_generate_probes**(value: GenerateProbes)
- GenerateProbes **get_generate_probes**()

The level of subdivision to use when automatically generating LightmapProbes for dynamic object lighting. Higher values result in more accurate indirect lighting on dynamic objects, at the cost of longer bake times and larger file sizes.

Note: Automatically generated LightmapProbes are not visible as nodes in the Scene tree dock, and cannot be modified this way after they are generated.

Note: Regardless of generate_probes_subdiv, direct lighting on dynamic objects is always applied using Light3D nodes in real-time.

---

bool **interior** = `false` 🔗

- void **set_interior**(value: bool)
- bool **is_interior**()

If `true`, ignore environment lighting when baking lightmaps.

LightmapGIData **light_data** 🔗

- void **set_light_data**(value: LightmapGIData)
- LightmapGIData **get_light_data**()

The LightmapGIData associated to this **LightmapGI** node. This resource is automatically created after baking, and is not meant to be created manually.

---

int **max_texture_size** = `16384` 🔗

- void **set_max_texture_size**(value: int)
- int **get_max_texture_size**()

The maximum texture size for the generated texture atlas. Higher values will result in fewer slices being generated, but may not work on all hardware as a result of hardware limitations on texture sizes. Leave max_texture_size at its default value of `16384` if unsure.

BakeQuality **quality** = `1` 🔗

- void **set_bake_quality**(value: BakeQuality)
- BakeQuality **get_bake_quality**()

The quality preset to use when baking lightmaps. This affects bake times, but output file sizes remain mostly identical across quality levels.

To further speed up bake times, decrease bounces, disable use_denoiser and increase the lightmap texel size on 3D scenes in the Import doc.

---

float **texel_scale** = `1.0` 🔗

- void **set_texel_scale**(value: float)
- float **get_texel_scale**()

Scales the lightmap texel density of all meshes for the current bake. This is a multiplier that builds upon the existing lightmap texel size defined in each imported 3D scene, along with the per-mesh density multiplier (which is designed to be used when the same mesh is used at different scales). Lower values will result in faster bake times.

---

bool **use_denoiser** = `true` 🔗

- void **set_use_denoiser**(value: bool)
- bool **is_using_denoiser**()

If `true`, uses a GPU-based denoising algorithm on the generated lightmap. This eliminates most noise within the generated lightmap at the cost of longer bake times. File sizes are generally not impacted significantly by the use of a denoiser, although lossless compression may do a better job at compressing a denoised image.

---

bool **use_texture_for_bounces** = `true` 🔗

- void **set_use_texture_for_bounces**(value: bool)

- bool **is_using_texture_for_bounces**()

If `true`, a texture with the lighting information will be generated to speed up the generation of indirect lighting at the cost of some accuracy. The geometry might exhibit extra light leak artifacts when using low resolution lightmaps or UVs that stretch the lightmap significantly across surfaces. Leave use_texture_for_bounces at its default value of `true` if unsure.

Note: use_texture_for_bounces only has an effect if bounces is set to a value greater than or equal to `1`.

LightmapProbe

Inherits: Node3D **<** Node **<** Object

Represents a single manually placed probe for dynamic object lighting with LightmapGI.

Description

LightmapProbe represents the position of a single manually placed probe for dynamic object lighting with LightmapGI. Lightmap probes affect the lighting of GeometryInstance3D-derived nodes that have their GeometryInstance3D.gi_mode set to GeometryInstance3D.GI_MODE_DYNAMIC.

Typically, LightmapGI probes are placed automatically by setting LightmapGI.generate_probes_subdiv to a value other than LightmapGI.GENERATE_PROBES_DISABLED. By creating **LightmapProbe** nodes before baking lightmaps, you can add more probes in specific areas for greater detail, or disable automatic generation and rely only on manually placed probes instead.

Note: LightmapProbe nodes that are placed after baking lightmaps are ignored by dynamic objects. You must bake lightmaps again after creating or modifying **LightmapProbe**s for the probes to be effective.

LightOccluder2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Occludes light cast by a Light2D, casting shadows.

Description

Occludes light cast by a Light2D, casting shadows. The LightOccluder2D must be provided with an OccluderPolygon2D in order for the shadow to be computed.

Tutorials

- 2D lights and shadows

Properties

OccluderPolygon2D	occluder	
int	occluder_light_mask	`1`
bool	sdf_collision	`true`

Property Descriptions

OccluderPolygon2D **occluder** 🔗

- void **set_occluder_polygon**(value: OccluderPolygon2D)
- OccluderPolygon2D **get_occluder_polygon**()

The OccluderPolygon2D used to compute the shadow.

---

int **occluder_light_mask** = `1` 🔗

- void **set_occluder_light_mask**(value: int)
- int **get_occluder_light_mask**()

The LightOccluder2D's occluder light mask. The LightOccluder2D will cast shadows only from Light2D(s) that have the same light mask(s).

---

bool **sdf_collision** = `true` 🔗

- void **set_as_sdf_collision**(value: bool)
- bool **is_set_as_sdf_collision**()

If enabled, the occluder will be part of a real-time generated signed distance field that can be used in custom shaders.

Line2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A 2D polyline that can optionally be textured.

Description

This node draws a 2D polyline, i.e. a shape consisting of several points connected by segments. **Line2D** is not a mathematical polyline, i.e. the segments are not infinitely thin. It is intended for rendering and it can be colored and optionally textured.

Warning: Certain configurations may be impossible to draw nicely, such as very sharp angles. In these situations, the node uses fallback drawing logic to look decent.

Note: **Line2D** is drawn using a 2D mesh.

Tutorials

- Matrix Transform Demo [https://godotengine.org/asset-library/asset/2787]
- 2.5D Game Demo [https://godotengine.org/asset-library/asset/2783]

Properties

bool	antialiased	`false`

LineCapMode	begin_cap_mode	`0`
bool	closed	`false`
Color	default_color	`Color(1, 1, 1, 1)`
LineCapMode	end_cap_mode	`0`
Gradient	gradient	
LineJointMode	joint_mode	`0`
PackedVector2Array	points	`PackedVector2Array()`
int	round_precision	`8`
float	sharp_limit	`2.0`
Texture2D	texture	
LineTextureMode	texture_mode	`0`
float	width	`10.0`
Curve	width_curve	

Methods

void	add_point(position: Vector2, index: int = -1)
void	clear_points()
int	get_point_count() const
Vector2	get_point_position(index: int) const
void	remove_point(index: int)
void	set_point_position(index: int, position: Vector2)

Enumerations

enum **LineJointMode**: 🔗

LineJointMode **LINE_JOINT_SHARP** = `0`

Makes the polyline's joints pointy, connecting the sides of the two segments by extending them until they intersect. If the rotation of a joint is too big (based on sharp_limit), the joint falls back to LINE_JOINT_BEVEL to prevent very long miters.

LineJointMode **LINE_JOINT_BEVEL** = `1`

Makes the polyline's joints bevelled/chamfered, connecting the sides of the two segments with a simple line.

LineJointMode **LINE_JOINT_ROUND** = `2`

Makes the polyline's joints rounded, connecting the sides of the two segments with an arc. The detail of this arc depends on round_precision.

---

enum **LineCapMode**: 🔗

LineCapMode **LINE_CAP_NONE** = `0`

Draws no line cap.

LineCapMode **LINE_CAP_BOX** = `1`

Draws the line cap as a box, slightly extending the first/last segment.

LineCapMode **LINE_CAP_ROUND** = `2`

Draws the line cap as a semicircle attached to the first/last segment.

---

enum **LineTextureMode**: 🔗

LineTextureMode **LINE_TEXTURE_NONE** = `0`

Takes the left pixels of the texture and renders them over the whole polyline.

LineTextureMode **LINE_TEXTURE_TILE** = `1`

Tiles the texture over the polyline. CanvasItem.texture_repeat of the **Line2D** node must be CanvasItem.TEXTURE_REPEAT_ENABLED or

CanvasItem.TEXTURE_REPEAT_MIRROR for it to work properly.

LineTextureMode **LINE_TEXTURE_STRETCH** = `2`

Stretches the texture across the polyline. CanvasItem.texture_repeat of the **Line2D** node must be CanvasItem.TEXTURE_REPEAT_DISABLED for best results.

Property Descriptions

bool **antialiased** = `false`

- void **set_antialiased**(value: bool)
- bool **get_antialiased**()

If `true`, the polyline's border will be anti-aliased.

Note: Line2D is not accelerated by batching when being anti-aliased.

LineCapMode **begin_cap_mode** = `0`

- void **set_begin_cap_mode**(value: LineCapMode)
- LineCapMode **get_begin_cap_mode**()

The style of the beginning of the polyline, if closed is `false`. Use LineCapMode constants.

---

bool **closed** = `false`

- void **set_closed**(value: bool)
- bool **is_closed**()

If `true` and the polyline has more than 2 points, the last point and the first one will be connected by a segment.

Note: The shape of the closing segment is not guaranteed to be seamless if a width_curve is provided.

Note: The joint between the closing segment and the first segment is drawn first and it samples the gradient and the width_curve at the beginning. This is an implementation detail that might change in a future version.

Color **default_color** = `Color(1, 1, 1, 1)`

- void **set_default_color**(value: Color)
- Color **get_default_color**()

The color of the polyline. Will not be used if a gradient is set.

LineCapMode **end_cap_mode** = `0`

- void **set_end_cap_mode**(value: LineCapMode)
- LineCapMode **get_end_cap_mode**()

The style of the end of the polyline, if closed is `false`. Use LineCapMode constants.

Gradient **gradient**

- void **set_gradient**(value: Gradient)
- Gradient **get_gradient**()

The gradient is drawn through the whole line from start to finish. The default_color will not be used if this property is set.

LineJointMode **joint_mode** = `0`

- void **set_joint_mode**(value: LineJointMode)
- LineJointMode **get_joint_mode**()

The style of the connections between segments of the polyline. Use LineJointMode constants.

PackedVector2Array **points** = `PackedVector2Array()`

- void **set_points**(value: PackedVector2Array)
- PackedVector2Array **get_points**()

The points of the polyline, interpreted in local 2D coordinates. Segments are drawn between the adjacent points in this array.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

---

int **round_precision** = `8`

- void **set_round_precision**(value: int)
- int **get_round_precision**()

The smoothness used for rounded joints and caps. Higher values result in smoother corners, but are more demanding to render and update.

---

float **sharp_limit** = `2.0`

- void **set_sharp_limit**(value: float)
- float **get_sharp_limit**()

Determines the miter limit of the polyline. Normally, when joint_mode is set to LINE_JOINT_SHARP, sharp angles fall back to using the logic of LINE_JOINT_BEVEL joints to prevent very long miters. Higher values of this property mean that the fallback to a bevel joint will happen at sharper angles.

Texture2D **texture** 🔗

- void **set_texture**(value: Texture2D)
- Texture2D **get_texture**()

The texture used for the polyline. Uses texture_mode for drawing style.

LineTextureMode **texture_mode** = `0` 🔗

- void **set_texture_mode**(value: LineTextureMode)
- LineTextureMode **get_texture_mode**()

The style to render the texture of the polyline. Use LineTextureMode constants.

---

float **width** = `10.0` 🔗

- void **set_width**(value: float)
- float **get_width**()

The polyline's width.

Curve **width_curve** 🔗

- void **set_curve**(value: Curve)
- Curve **get_curve**()

The polyline's width curve. The width of the polyline over its length will be equivalent to the value of the width curve over its domain.

Method Descriptions

void **add_point**(position: Vector2, index: int = -1) 🔗

Adds a point with the specified `position` relative to the polyline's own position. If no `index` is provided, the new point will be added to the end of the points array.

If `index` is given, the new point is inserted before the existing point identified by index `index`. The indices of the points after the new point get increased by 1. The provided `index` must not exceed the number of existing points in the polyline. See get_point_count.

void **clear_points**() 🔗

Removes all points from the polyline, making it empty.

int **get_point_count**() const 🔗

Returns the number of points in the polyline.

Vector2 **get_point_position**(index: int) const 🔗

Returns the position of the point at index `index`.

void **remove_point**(index: int) 🔗

Removes the point at index `index` from the polyline.

---

void **set_point_position**(index: int, position: Vector2)

Overwrites the position of the point at the given `index` with the supplied `position`.

LineEdit

Inherits: Control **<** CanvasItem **<** Node **<** Object

An input field for single-line text.

Description

LineEdit provides an input field for editing a single line of text. It features many built-in shortcuts that are always available (`Ctrl` here maps to `Cmd` on macOS):

- `Ctrl + C`: Copy
- `Ctrl + X`: Cut
- `Ctrl + V` or `Ctrl + Y`: Paste/"yank"
- `Ctrl + Z`: Undo
- `Ctrl + ~`: Swap input direction.
- `Ctrl + Shift + Z`: Redo
- `Ctrl + U`: Delete text from the caret position to the beginning of the line
- `Ctrl + K`: Delete text from the caret position to the end of the line
- `Ctrl + A`: Select all text
- `Up Arrow`/`Down Arrow`: Move the caret to the beginning/end of the line

On macOS, some extra keyboard shortcuts are available:

- `Cmd + F`: Same as `Right Arrow`, move the caret one character right
- `Cmd + B`: Same as `Left Arrow`, move the caret one character left
- `Cmd + P`: Same as `Up Arrow`, move the caret to the previous line
- `Cmd + N`: Same as `Down Arrow`, move the caret to the next line
- `Cmd + D`: Same as `Delete`, delete the character on the right side of caret

- `Cmd + H`: Same as `Backspace`, delete the character on the left side of the caret
- `Cmd + A`: Same as `Home`, move the caret to the beginning of the line
- `Cmd + E`: Same as `End`, move the caret to the end of the line
- `Cmd + Left Arrow`: Same as `Home`, move the caret to the beginning of the line
- `Cmd + Right Arrow`: Same as `End`, move the caret to the end of the line

Properties

HorizontalAlignment	alignment	`0`
bool	caret_blink	`false`
float	caret_blink_interval	`0.65`
int	caret_column	`0`
bool	caret_force_displayed	`false`
bool	caret_mid_grapheme	`false`
bool	clear_button_enabled	`false`
bool	context_menu_enabled	`true`
bool	deselect_on_focus_loss_enabled	`true`

bool	drag_and_drop_selection_enabled	`true`
bool	draw_control_chars	`false`
bool	editable	`true`
bool	expand_to_text_length	`false`
bool	flat	`false`
FocusMode	focus_mode	`2` (overrides Control)
String	language	`""`
int	max_length	`0`
bool	middle_mouse_paste_enabled	`true`
CursorShape	mouse_default_cursor_shape	`1` (overrides Control)
String	placeholder_text	`""`
Texture2D	right_icon	

bool	secret	`false`
String	secret_character	`"•"`
bool	select_all_on_focus	`false`
bool	selecting_enabled	`true`
bool	shortcut_keys_enabled	`true`
StructuredTextParser	structured_text_bidi_override	`0`
Array	structured_text_bidi_override_options	`[]`
String	text	`""`
TextDirection	text_direction	`0`
bool	virtual_keyboard_enabled	`true`
VirtualKeyboardType	virtual_keyboard_type	`0`

Methods

void	clear()

void	delete_char_at_caret()
void	delete_text(from_column: int, to_column: int)
void	deselect()
PopupMenu	get_menu() const
float	get_scroll_offset() const
String	get_selected_text()
int	get_selection_from_column() const
int	get_selection_to_column() const
bool	has_selection() const
void	insert_text_at_caret(text: String)
bool	is_menu_visible() const
void	menu_option(option: int)
void	select(from: int = 0, to: int = -1)

void	select_all()

Theme Properties

Color	caret_color	`Color(0.95, 0.95, 0.95, 1)`
Color	clear_button_color	`Color(0.875, 0.875, 0.875, 1)`
Color	clear_button_color_pressed	`Color(1, 1, 1, 1)`
Color	font_color	`Color(0.875, 0.875, 0.875, 1)`
Color	font_outline_color	`Color(0, 0, 0, 1)`
Color	font_placeholder_color	`Color(0.875, 0.875, 0.875, 0.6)`
Color	font_selected_color	`Color(1, 1, 1, 1)`
Color	font_uneditable_color	`Color(0.875, 0.875, 0.875, 0.5)`
Color	selection_color	`Color(0.5, 0.5, 0.5, 1)`

int	caret_width	`1`
int	minimum_character_width	`4`
int	outline_size	`0`
Font	font	
int	font_size	
Texture2D	clear	
StyleBox	focus	
StyleBox	normal	
StyleBox	read_only	

Signals

text_change_rejected(rejected_substring: String) 🔗

Emitted when appending text that overflows the max_length. The appended text is truncated to fit max_length, and the part that couldn't fit is passed as the `rejected_substring` argument.

---

text_changed(new_text: String) 🔗

Emitted when the text changes.

---

text_submitted(new_text: String) 🔗

Emitted when the user presses @GlobalScope.KEY_ENTER on the **LineEdit**.

Enumerations

enum **MenuItems**: 🔗

MenuItems **MENU_CUT** = `0`

Cuts (copies and clears) the selected text.

MenuItems **MENU_COPY** = `1`

Copies the selected text.

MenuItems **MENU_PASTE** = `2`

Pastes the clipboard text over the selected text (or at the caret's position).

Non-printable escape characters are automatically stripped from the OS clipboard via String.strip_escapes.

MenuItems **MENU_CLEAR** = `3`

Erases the whole **LineEdit** text.

MenuItems **MENU_SELECT_ALL** = `4`

Selects the whole **LineEdit** text.

MenuItems **MENU_UNDO** = `5`

Undoes the previous action.

MenuItems **MENU_REDO** = `6`

Reverse the last undo action.

MenuItems **MENU_SUBMENU_TEXT_DIR** = `7`

ID of "Text Writing Direction" submenu.

MenuItems **MENU_DIR_INHERITED** = `8`

Sets text direction to inherited.

MenuItems **MENU_DIR_AUTO** = `9`

Sets text direction to automatic.

MenuItems **MENU_DIR_LTR** = `10`

Sets text direction to left-to-right.

MenuItems **MENU_DIR_RTL** = `11`

Sets text direction to right-to-left.

MenuItems **MENU_DISPLAY_UCC** = `12`

Toggles control character display.

MenuItems **MENU_SUBMENU_INSERT_UCC** = `13`

ID of "Insert Control Character" submenu.

MenuItems **MENU_INSERT_LRM** = `14`

Inserts left-to-right mark (LRM) character.

MenuItems **MENU_INSERT_RLM** = `15`

Inserts right-to-left mark (RLM) character.

MenuItems **MENU_INSERT_LRE** = `16`

Inserts start of left-to-right embedding (LRE) character.

MenuItems **MENU_INSERT_RLE** = `17`

Inserts start of right-to-left embedding (RLE) character.

MenuItems **MENU_INSERT_LRO** = `18`

Inserts start of left-to-right override (LRO) character.

MenuItems **MENU_INSERT_RLO** = `19`

Inserts start of right-to-left override (RLO) character.

MenuItems **MENU_INSERT_PDF** = `20`

Inserts pop direction formatting (PDF) character.

MenuItems **MENU_INSERT_ALM** = `21`

Inserts Arabic letter mark (ALM) character.

MenuItems **MENU_INSERT_LRI** = `22`

Inserts left-to-right isolate (LRI) character.

MenuItems **MENU_INSERT_RLI** = `23`

Inserts right-to-left isolate (RLI) character.

MenuItems **MENU_INSERT_FSI** = `24`

Inserts first strong isolate (FSI) character.

MenuItems **MENU_INSERT_PDI** = `25`

Inserts pop direction isolate (PDI) character.

MenuItems **MENU_INSERT_ZWJ** = `26`

Inserts zero width joiner (ZWJ) character.

MenuItems **MENU_INSERT_ZWNJ** = `27`

Inserts zero width non-joiner (ZWNJ) character.

MenuItems **MENU_INSERT_WJ** = `28`

Inserts word joiner (WJ) character.

MenuItems **MENU_INSERT_SHY** = `29`

Inserts soft hyphen (SHY) character.

MenuItems **MENU_MAX** = `30`

Represents the size of the MenuItems enum.

enum **VirtualKeyboardType**: 🔗

VirtualKeyboardType **KEYBOARD_TYPE_DEFAULT** = `0`

Default text virtual keyboard.

VirtualKeyboardType **KEYBOARD_TYPE_MULTILINE** = `1`

Multiline virtual keyboard.

VirtualKeyboardType **KEYBOARD_TYPE_NUMBER** = `2`

Virtual number keypad, useful for PIN entry.

VirtualKeyboardType **KEYBOARD_TYPE_NUMBER_DECIMAL** = `3`

Virtual number keypad, useful for entering fractional numbers.

VirtualKeyboardType **KEYBOARD_TYPE_PHONE** = `4`

Virtual phone number keypad.

VirtualKeyboardType **KEYBOARD_TYPE_EMAIL_ADDRESS** = `5`

Virtual keyboard with additional keys to assist with typing email addresses.

VirtualKeyboardType **KEYBOARD_TYPE_PASSWORD** = `6`

Virtual keyboard for entering a password. On most platforms, this should disable autocomplete and autocapitalization.

Note: This is not supported on Web. Instead, this behaves identically to KEYBOARD_TYPE_DEFAULT.

VirtualKeyboardType **KEYBOARD_TYPE_URL** = `7`

Virtual keyboard with additional keys to assist with typing URLs.

Property Descriptions

HorizontalAlignment **alignment** = `0` 🔗

- void **set_horizontal_alignment**(value: HorizontalAlignment)
- HorizontalAlignment **get_horizontal_alignment**()

Text alignment as defined in the HorizontalAlignment enum.

---

bool **caret_blink** = `false` 🔗

- void **set_caret_blink_enabled**(value: bool)
- bool **is_caret_blink_enabled**()

If `true`, makes the caret blink.

---

float **caret_blink_interval** = `0.65`

- void **set_caret_blink_interval**(value: float)
- float **get_caret_blink_interval**()

The interval at which the caret blinks (in seconds).

---

int **caret_column** = `0`

- void **set_caret_column**(value: int)
- int **get_caret_column**()

The caret's column position inside the **LineEdit**. When set, the text may scroll to accommodate it.

---

bool **caret_force_displayed** = `false`

- void **set_caret_force_displayed**(value: bool)
- bool **is_caret_force_displayed**()

If `true`, the **LineEdit** will always show the caret, even if focus is lost.

---

bool **caret_mid_grapheme** = `false`

- void **set_caret_mid_grapheme_enabled**(value: bool)
- bool **is_caret_mid_grapheme_enabled**()

Allow moving caret, selecting and removing the individual composite character components.

Note: `Backspace` is always removing individual composite character components.

---

bool **clear_button_enabled** = `false`

- void **set_clear_button_enabled**(value: bool)
- bool **is_clear_button_enabled**()

If `true`, the **LineEdit** will show a clear button if text is not empty, which can be used to clear the text quickly.

---

bool **context_menu_enabled** = `true`

- void **set_context_menu_enabled**(value: bool)
- bool **is_context_menu_enabled**()

If `true`, the context menu will appear when right-clicked.

---

bool **deselect_on_focus_loss_enabled** = `true`

- void **set_deselect_on_focus_loss_enabled**(value: bool)
- bool **is_deselect_on_focus_loss_enabled**()

If `true`, the selected text will be deselected when focus is lost.

---

bool **drag_and_drop_selection_enabled** = `true`

- void **set_drag_and_drop_selection_enabled**(value: bool)
- bool **is_drag_and_drop_selection_enabled**()

If `true`, allow drag and drop of selected text.

---

bool **draw_control_chars** = `false`

- void **set_draw_control_chars**(value: bool)
- bool **get_draw_control_chars**()

If `true`, control characters are displayed.

---

bool **editable** = `true`

- void **set_editable**(value: bool)
- bool **is_editable**()

If `false`, existing text cannot be modified and new text cannot be added.

---

bool **expand_to_text_length** = `false`

- void **set_expand_to_text_length_enabled**(value: bool)
- bool **is_expand_to_text_length_enabled**()

If `true`, the **LineEdit** width will increase to stay longer than the text. It will **not** compress if the text is shortened.

---

bool **flat** = `false`

- void **set_flat**(value: bool)
- bool **is_flat**()

If `true`, the **LineEdit** doesn't display decoration.

String **language** = `""`

- void **set_language**(value: String)
- String **get_language**()

Language code used for line-breaking and text shaping algorithms. If left empty, current locale is used instead.

---

int **max_length** = `0`

- void **set_max_length**(value: int)

- int **get_max_length**()

Maximum number of characters that can be entered inside the **LineEdit**. If `0`, there is no limit.

When a limit is defined, characters that would exceed max_length are truncated. This happens both for existing text contents when setting the max length, or for new text inserted in the **LineEdit**, including pasting. If any input text is truncated, the text_change_rejected signal is emitted with the truncated substring as parameter.

Example:

GDScript

```
text = "Hello world"
max_length = 5
# `text` becomes "Hello".
max_length = 10
text += " goodbye"
# `text` becomes "Hello good".
# `text_change_rejected` is emitted with "bye" as parameter.
```

C#

```
Text = "Hello world";
MaxLength = 5;
// `Text` becomes "Hello".
MaxLength = 10;
Text += " goodbye";
// `Text` becomes "Hello good".
// `text_change_rejected` is emitted with "bye" as parameter.
```

---

bool **middle_mouse_paste_enabled** = `true`

- void **set_middle_mouse_paste_enabled**(value: bool)
- bool **is_middle_mouse_paste_enabled**()

If `false`, using middle mouse button to paste clipboard will be disabled.

Note: This method is only implemented on Linux.

String **placeholder_text** = `""` 🔗

- void **set_placeholder**(value: String)
- String **get_placeholder**()

Text shown when the **LineEdit** is empty. It is **not** the **LineEdit**'s default value (see text).

Texture2D **right_icon** 🔗

- void **set_right_icon**(value: Texture2D)
- Texture2D **get_right_icon**()

Sets the icon that will appear in the right end of the **LineEdit** if there's no text, or always, if clear_button_enabled is set to `false`.

---

bool **secret** = `false` 🔗

- void **set_secret**(value: bool)
- bool **is_secret**()

If `true`, every character is replaced with the secret character (see secret_character).

String **secret_character** = `"•"` 🔗

- void **set_secret_character**(value: String)
- String **get_secret_character**()

The character to use to mask secret input. Only a single character can be used as the secret character. If it is longer than one character, only the first one will be used. If it is empty, a space will be used instead.

---

bool **select_all_on_focus** = `false`

- void **set_select_all_on_focus**(value: bool)
- bool **is_select_all_on_focus**()

If `true`, the **LineEdit** will select the whole text when it gains focus.

bool **selecting_enabled** = `true`

- void **set_selecting_enabled**(value: bool)
- bool **is_selecting_enabled**()

If `false`, it's impossible to select the text using mouse nor keyboard.

bool **shortcut_keys_enabled** = `true`

- void **set_shortcut_keys_enabled**(value: bool)
- bool **is_shortcut_keys_enabled**()

If `false`, using shortcuts will be disabled.

StructuredTextParser **structured_text_bidi_override** = `0`

- void **set_structured_text_bidi_override**(value: StructuredTextParser)
- StructuredTextParser **get_structured_text_bidi_override**()

Set BiDi algorithm override for the structured text.

Array **structured_text_bidi_override_options** = `[]`

- void **set_structured_text_bidi_override_options**(value: Array)
- Array **get_structured_text_bidi_override_options**()

Set additional options for BiDi override.

String **text** = `""`

- void **set_text**(value: String)
- String **get_text**()

String value of the **LineEdit**.

Note: Changing text using this property won't emit the text_changed signal.

TextDirection **text_direction** = `0`

- void **set_text_direction**(value: TextDirection)
- TextDirection **get_text_direction**()

Base text writing direction.

---

bool **virtual_keyboard_enabled** = `true`

- void **set_virtual_keyboard_enabled**(value: bool)
- bool **is_virtual_keyboard_enabled**()

If `true`, the native virtual keyboard is shown when focused on platforms that support it.

VirtualKeyboardType **virtual_keyboard_type** = `0`

- void **set_virtual_keyboard_type**(value: VirtualKeyboardType)
- VirtualKeyboardType **get_virtual_keyboard_type**()

Specifies the type of virtual keyboard to show.

Method Descriptions

void **clear**() 🔗

Erases the **LineEdit**'s text.

---

void **delete_char_at_caret**() 🔗

Deletes one character at the caret's current position (equivalent to pressing `Delete`).

---

void **delete_text**(from_column: int, to_column: int) 🔗

Deletes a section of the text going from position `from_column` to `to_column`. Both parameters should be within the text's length.

---

void **deselect**() 🔗

Clears the current selection.

PopupMenu **get_menu**() const 🔗

Returns the PopupMenu of this **LineEdit**. By default, this menu is displayed when right-clicking on the **LineEdit**.

You can add custom menu items or remove standard ones. Make sure your IDs don't conflict with the standard ones (see MenuItems). For example:

GDScript

```
func _ready():
    var menu = get_menu()
    # Remove all items after "Redo".
    menu.item_count = menu.get_item_index(MENU_REDO) + 1
    # Add custom items.
    menu.add_separator()
    menu.add_item("Insert Date", MENU_MAX + 1)
    # Connect callback.
    menu.id_pressed.connect(_on_item_pressed)

func _on_item_pressed(id):
    if id == MENU_MAX + 1:
        insert_text_at_caret(Time.get_date_string_from_system())
```

C#

```
public override void _Ready()
{
    var menu = GetMenu();
    // Remove all items after "Redo".
    menu.ItemCount = menu.GetItemIndex(LineEdit.MenuItems.Redo) +
1;
    // Add custom items.
    menu.AddSeparator();
    menu.AddItem("Insert Date", LineEdit.MenuItems.Max + 1);
    // Add event handler.
    menu.IdPressed += OnItemPressed;
}

public void OnItemPressed(int id)
{
    if (id == LineEdit.MenuItems.Max + 1)
    {
        InsertTextAtCaret(Time.GetDateStringFromSystem());
    }
}
```

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their Window.visible property.

---

float **get_scroll_offset**() const 🔗

Returns the scroll offset due to caret_column, as a number of characters.

String **get_selected_text**()

Returns the text inside the selection.

---

int **get_selection_from_column**() const

Returns the selection begin column.

---

int **get_selection_to_column**() const

Returns the selection end column.

---

bool **has_selection**() const

Returns `true` if the user has selected text.

---

void **insert_text_at_caret**(text: String)

Inserts `text` at the caret. If the resulting value is longer than max_length, nothing happens.

---

bool **is_menu_visible**() const

Returns whether the menu is visible. Use this instead of `get_menu().visible` to improve performance (so the creation of the menu is avoided).

---

void **menu_option**(option: int)

Executes a given action as defined in the MenuItems enum.

---

void **select**(from: int = 0, to: int = -1) 🔗

Selects characters inside **LineEdit** between `from` and `to`. By default, `from` is at the beginning and `to` at the end.

GDScript

```
text = "Welcome"
select() # Will select "Welcome".
select(4) # Will select "ome".
select(2, 5) # Will select "lco".
```

C#

```
Text = "Welcome";
Select(); // Will select "Welcome".
Select(4); // Will select "ome".
Select(2, 5); // Will select "lco".
```

---

void **select_all**() 🔗

Selects the whole String.

Theme Property Descriptions

Color **caret_color** = `Color(0.95, 0.95, 0.95, 1)` 🔗

Color of the **LineEdit**'s caret (text cursor). This can be set to a fully transparent color to hide the caret entirely.

Color **clear_button_color** = `Color(0.875, 0.875, 0.875, 1)` 🔗

Color used as default tint for the clear button.

Color **clear_button_color_pressed** = `Color(1, 1, 1, 1)` 🔗

Color used for the clear button when it's pressed.

Color **font_color** = `Color(0.875, 0.875, 0.875, 1)`

Default font color.

Color **font_outline_color** = `Color(0, 0, 0, 1)`

The tint of text outline of the **LineEdit**.

Color **font_placeholder_color** = `Color(0.875, 0.875, 0.875, 0.6)`

Font color for placeholder_text.

Color **font_selected_color** = `Color(1, 1, 1, 1)`

Font color for selected text (inside the selection rectangle).

Color **font_uneditable_color** = `Color(0.875, 0.875, 0.875, 0.5)`

Font color when editing is disabled.

Color **selection_color** = `Color(0.5, 0.5, 0.5, 1)`

Color of the selection rectangle.

int **caret_width** = `1`

The caret's width in pixels. Greater values can be used to improve accessibility by ensuring the caret is easily visible, or to ensure consistency with a large font size.

---

int **minimum_character_width** = `4`

Minimum horizontal space for the text (not counting the clear button and content margins). This value is measured in count of 'M' characters (i.e. this number of 'M' characters can be displayed without scrolling).

---

int **outline_size** = `0`

The size of the text outline.

Note: If using a font with FontFile.multichannel_signed_distance_field enabled, its FontFile.msdf_pixel_range must be set to at least *twice* the value of outline_size for outline rendering to look correct. Otherwise, the outline may appear to be cut off earlier than intended.

Font **font**

Font used for the text.

---

int **font_size**

Font size of the **LineEdit**'s text.

Texture2D **clear**

Texture for the clear button. See clear_button_enabled.

StyleBox **focus**

Background used when **LineEdit** has GUI focus. The focus StyleBox is displayed *over* the base StyleBox, so a partially transparent StyleBox should be used to ensure the base StyleBox remains visible. A StyleBox that represents an outline or an underline works well for this purpose. To disable the focus visual effect, assign a StyleBoxEmpty resource. Note that disabling the focus visual effect will harm keyboard/controller navigation usability, so this is not recommended for accessibility reasons.

StyleBox **normal** 🔗

Default background for the **LineEdit**.

StyleBox **read_only** 🔗

Background used when **LineEdit** is in read-only mode (editable is set to `false`).

LinkButton

Inherits: BaseButton **<** Control **<** CanvasItem **<** Node **<** Object

A button that represents a link.

Description

A button that represents a link. This type of button is primarily used for interactions that cause a context change (like linking to a web page).

See also BaseButton which contains common properties and methods associated with this node.

Properties

FocusMode	focus_mode	`0` (overrides Control)
String	language	`""`
CursorShape	mouse_default_cursor_shape	`2` (overrides Control)
StructuredTextParser	structured_text_bidi_override	`0`
Array	structured_text_bidi_override_options	`[]`

String	text	""
TextDirection	text_direction	0
UnderlineMode	underline	0
String	uri	""

Theme Properties

Color	font_color	`Color(0.875, 0.875, 0.875, 1)`
Color	font_disabled_color	`Color(0, 0, 0, 1)`
Color	font_focus_color	`Color(0.95, 0.95, 0.95, 1)`
Color	font_hover_color	`Color(0.95, 0.95, 0.95, 1)`
Color	font_hover_pressed_color	`Color(0, 0, 0, 1)`
Color	font_outline_color	`Color(0, 0, 0, 1)`
Color	font_pressed_color	`Color(1, 1, 1, 1)`
int	outline_size	0

int	underline_spacing	2
Font	font	
int	font_size	
StyleBox	focus	

Enumerations

enum **UnderlineMode**: 🔗

UnderlineMode **UNDERLINE_MODE_ALWAYS** = `0`

The LinkButton will always show an underline at the bottom of its text.

UnderlineMode **UNDERLINE_MODE_ON_HOVER** = `1`

The LinkButton will show an underline at the bottom of its text when the mouse cursor is over it.

UnderlineMode **UNDERLINE_MODE_NEVER** = `2`

The LinkButton will never show an underline at the bottom of its text.

Property Descriptions

String **language** = `""` 🔗

- void **set_language**(value: String)
- String **get_language**()

Language code used for line-breaking and text shaping algorithms, if left empty current locale is used instead.

StructuredTextParser **structured_text_bidi_override** = `0`

- void **set_structured_text_bidi_override**(value: StructuredTextParser)
- StructuredTextParser **get_structured_text_bidi_override**()

Set BiDi algorithm override for the structured text.

Array **structured_text_bidi_override_options** = `[]`

- void **set_structured_text_bidi_override_options**(value: Array)
- Array **get_structured_text_bidi_override_options**()

Set additional options for BiDi override.

String **text** = `""`

- void **set_text**(value: String)
- String **get_text**()

The button's text that will be displayed inside the button's area.

TextDirection **text_direction** = `0`

- void **set_text_direction**(value: TextDirection)
- TextDirection **get_text_direction**()

Base text writing direction.

UnderlineMode **underline** = `0` 🔗

- void **set_underline_mode**(value: UnderlineMode)
- UnderlineMode **get_underline_mode**()

The underline mode to use for the text. See UnderlineMode for the available modes.

String **uri** = `""` 🔗

- void **set_uri**(value: String)
- String **get_uri**()

The URI [https://en.wikipedia.org/wiki/Uniform_Resource_Identifier] for this **LinkButton**. If set to a valid URI, pressing the button opens the URI using the operating system's default program for the protocol (via OS.shell_open). HTTP and HTTPS URLs open the default web browser.

Examples:

GDScript

```
uri = "https://godotengine.org"  # Opens the URL in the default web browser.
uri = "C:\SomeFolder"  # Opens the file explorer at the given path.
uri = "C:\SomeImage.png"  # Opens the given image in the default viewing app.
```

C#

```
Uri = "https://godotengine.org"; // Opens the URL in the default web browser.
Uri = "C:\SomeFolder"; // Opens the file explorer at the given path.
Uri = "C:\SomeImage.png"; // Opens the given image in the default viewing app.
```

Theme Property Descriptions

Color **font_color** = `Color(0.875, 0.875, 0.875, 1)` 🔗

Default text Color of the **LinkButton**.

Color **font_disabled_color** = `Color(0, 0, 0, 1)` 🔗

Text Color used when the **LinkButton** is disabled.

Color **font_focus_color** = `Color(0.95, 0.95, 0.95, 1)` 🔗

Text Color used when the **LinkButton** is focused. Only replaces the normal text color of the button. Disabled, hovered, and pressed states take precedence over this color.

Color **font_hover_color** = `Color(0.95, 0.95, 0.95, 1)` 🔗

Text Color used when the **LinkButton** is being hovered.

Color **font_hover_pressed_color** = `Color(0, 0, 0, 1)` 🔗

Text Color used when the **LinkButton** is being hovered and pressed.

Color **font_outline_color** = `Color(0, 0, 0, 1)` 🔗

The tint of text outline of the **LinkButton**.

Color **font_pressed_color** = `Color(1, 1, 1, 1)` 🔗

Text Color used when the **LinkButton** is being pressed.

---

int **outline_size** = `0` 🔗

The size of the text outline.

Note: If using a font with FontFile.multichannel_signed_distance_field enabled, its FontFile.msdf_pixel_range must be set to at least *twice* the value of outline_size for outline rendering to look correct. Otherwise, the outline may appear to be cut off earlier than intended.

---

int **underline_spacing** = `2` 🔗

The vertical space between the baseline of text and the underline.

Font **font** 🔗

Font of the **LinkButton**'s text.

---

int **font_size** 🔗

Font size of the **LinkButton**'s text.

StyleBox **focus** 🔗

StyleBox used when the **LinkButton** is focused. The focus StyleBox is displayed *over* the base StyleBox, so a partially transparent StyleBox should be used to ensure the base StyleBox remains visible. A StyleBox that represents an outline or an underline works well for this purpose. To disable the focus visual effect, assign a StyleBoxEmpty resource. Note that disabling the focus visual effect will harm keyboard/controller navigation usability, so this is not recommended for accessibility reasons.

MarginContainer

Inherits: Container **<** Control **<** CanvasItem **<** Node **<** Object

A container that keeps a margin around its child controls.

Description

MarginContainer adds an adjustable margin on each side of its child controls. The margins are added around all children, not around each individual one. To control the **MarginContainer**'s margins, use the `margin_*` theme properties listed below.

Note: The margin sizes are theme overrides, not normal properties. This is an example of how to change them in code:

GDScript

```
# This code sample assumes the current script is extending
MarginContainer.
var margin_value = 100
add_theme_constant_override("margin_top", margin_value)
add_theme_constant_override("margin_left", margin_value)
add_theme_constant_override("margin_bottom", margin_value)
add_theme_constant_override("margin_right", margin_value)
```

C#

```
// This code sample assumes the current script is extending
MarginContainer.
int marginValue = 100;
AddThemeConstantOverride("margin_top", marginValue);
AddThemeConstantOverride("margin_left", marginValue);
AddThemeConstantOverride("margin_bottom", marginValue);
AddThemeConstantOverride("margin_right", marginValue);
```

Tutorials

- Using Containers

Theme Properties

int	margin_bottom	0
int	margin_left	0
int	margin_right	0
int	margin_top	0

Theme Property Descriptions

int **margin_bottom** = 0 🔗

Offsets towards the inside direct children of the container by this amount of pixels from the bottom.

int **margin_left** = 0 🔗

Offsets towards the inside direct children of the container by this amount of pixels from the left.

int **margin_right** = 0 🔗

Offsets towards the inside direct children of the container by this amount of pixels from the right.

---

int **margin_top** = 0 🔗

Offsets towards the inside direct children of the container by this amount of pixels from the top.

Marker2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Generic 2D position hint for editing.

Description

Generic 2D position hint for editing. It's just like a plain Node2D, but it displays as a cross in the 2D editor at all times. You can set the cross' visual size by using the gizmo in the 2D editor while the node is selected.

Properties

float	gizmo_extents	`10.0`

Property Descriptions

float **gizmo_extents** = `10.0` 🔗

- void **set_gizmo_extents**(value: float)
- float **get_gizmo_extents**()

Size of the gizmo cross that appears in the editor.

Marker3D

Inherits: Node3D **<** Node **<** Object

Generic 3D position hint for editing.

Description

Generic 3D position hint for editing. It's just like a plain Node3D, but it displays as a cross in the 3D editor at all times.

Properties

float	gizmo_extents	`0.25`

Property Descriptions

float **gizmo_extents** = `0.25` 🔗

- void **set_gizmo_extents**(value: float)
- float **get_gizmo_extents**()

Size of the gizmo cross that appears in the editor.

MenuBar

Inherits: Control **<** CanvasItem **<** Node **<** Object

A horizontal menu bar that creates a MenuButton for each PopupMenu child.

Description

A horizontal menu bar that creates a MenuButton for each PopupMenu child. New items are created by adding PopupMenus to this node.

Properties

bool	flat	`false`
String	language	`""`
bool	prefer_global_menu	`true`
int	start_index	`-1`
bool	switch_on_hover	`true`

TextDirection	text_direction	0

Methods

int	get_menu_count() const
PopupMenu	get_menu_popup(menu: int) const
String	get_menu_title(menu: int) const
String	get_menu_tooltip(menu: int) const
bool	is_menu_disabled(menu: int) const
bool	is_menu_hidden(menu: int) const
bool	is_native_menu() const
void	set_disable_shortcuts(disabled: bool)
void	set_menu_disabled(menu: int, disabled: bool)
void	set_menu_hidden(menu: int, hidden: bool)

void	set_menu_title(menu: int, title: String)
void	set_menu_tooltip(menu: int, tooltip: String)

Theme Properties

Color	font_color	`Color(0.875, 0.875, 0.875, 1)`
Color	font_disabled_color	`Color(0.875, 0.875, 0.875, 0.5)`
Color	font_focus_color	`Color(0.95, 0.95, 0.95, 1)`
Color	font_hover_color	`Color(0.95, 0.95, 0.95, 1)`
Color	font_hover_pressed_color	`Color(1, 1, 1, 1)`
Color	font_outline_color	`Color(0, 0, 0, 1)`
Color	font_pressed_color	`Color(1, 1, 1, 1)`
int	h_separation	`4`

int	outline_size	0
Font	font	
int	font_size	
StyleBox	disabled	
StyleBox	disabled_mirrored	
StyleBox	hover	
StyleBox	hover_mirrored	
StyleBox	hover_pressed	
StyleBox	hover_pressed_mirrored	
StyleBox	normal	
StyleBox	normal_mirrored	
StyleBox	pressed	
StyleBox	pressed_mirrored	

Property Descriptions

bool **flat** = `false`

- void **set_flat**(value: bool)
- bool **is_flat**()

Flat **MenuBar** don't display item decoration.

String **language** = `""`

- void **set_language**(value: String)
- String **get_language**()

Language code used for line-breaking and text shaping algorithms, if left empty current locale is used instead.

---

bool **prefer_global_menu** = `true`

- void **set_prefer_global_menu**(value: bool)
- bool **is_prefer_global_menu**()

If `true`, **MenuBar** will use system global menu when supported.

---

int **start_index** = `-1`

- void **set_start_index**(value: int)
- int **get_start_index**()

Position in the global menu to insert first **MenuBar** item at.

---

bool **switch_on_hover** = `true` 🔗

- void **set_switch_on_hover**(value: bool)
- bool **is_switch_on_hover**()

If `true`, when the cursor hovers above menu item, it will close the current PopupMenu and open the other one.

TextDirection **text_direction** = `0` 🔗

- void **set_text_direction**(value: TextDirection)
- TextDirection **get_text_direction**()

Base text writing direction.

Method Descriptions

int **get_menu_count**() const 🔗

Returns number of menu items.

PopupMenu **get_menu_popup**(menu: int) const 🔗

Returns PopupMenu associated with menu item.

String **get_menu_title**(menu: int) const 🔗

Returns menu item title.

String **get_menu_tooltip**(menu: int) const 🔗

Returns menu item tooltip.

bool **is_menu_disabled**(menu: int) const

Returns `true`, if menu item is disabled.

bool **is_menu_hidden**(menu: int) const

Returns `true`, if menu item is hidden.

bool **is_native_menu**() const

Returns `true`, if system global menu is supported and used by this **MenuBar**.

void **set_disable_shortcuts**(disabled: bool)

If `true`, shortcuts are disabled and cannot be used to trigger the button.

void **set_menu_disabled**(menu: int, disabled: bool)

If `true`, menu item is disabled.

void **set_menu_hidden**(menu: int, hidden: bool)

If `true`, menu item is hidden.

void **set_menu_title**(menu: int, title: String)

Sets menu item title.

---

void **set_menu_tooltip**(menu: int, tooltip: String)

Sets menu item tooltip.

Theme Property Descriptions

Color **font_color** = `Color(0.875, 0.875, 0.875, 1)`

Default text Color of the menu item.

Color **font_disabled_color** = `Color(0.875, 0.875, 0.875, 0.5)`

Text Color used when the menu item is disabled.

Color **font_focus_color** = `Color(0.95, 0.95, 0.95, 1)`

Text Color used when the menu item is focused. Only replaces the normal text color of the menu item. Disabled, hovered, and pressed states take precedence over this color.

Color **font_hover_color** = `Color(0.95, 0.95, 0.95, 1)`

Text Color used when the menu item is being hovered.

Color **font_hover_pressed_color** = `Color(1, 1, 1, 1)`

Text Color used when the menu item is being hovered and pressed.

Color **font_outline_color** = `Color(0, 0, 0, 1)`

The tint of text outline of the menu item.

Color **font_pressed_color** = `Color(1, 1, 1, 1)`

Text Color used when the menu item is being pressed.

int **h_separation** = `4`

The horizontal space between menu items.

int **outline_size** = `0`

The size of the text outline.

Note: If using a font with FontFile.multichannel_signed_distance_field enabled, its FontFile.msdf_pixel_range must be set to at least *twice* the value of outline_size for outline rendering to look correct. Otherwise, the outline may appear to be cut off earlier than intended.

Font **font**

Font of the menu item's text.

int **font_size**

Font size of the menu item's text.

StyleBox **disabled**

StyleBox used when the menu item is disabled.

StyleBox **disabled_mirrored**

StyleBox used when the menu item is disabled (for right-to-left layouts).

StyleBox **hover**

StyleBox used when the menu item is being hovered.

StyleBox **hover_mirrored**

StyleBox used when the menu item is being hovered (for right-to-left layouts).

StyleBox **hover_pressed**

StyleBox used when the menu item is being pressed and hovered at the same time.

StyleBox **hover_pressed_mirrored**

StyleBox used when the menu item is being pressed and hovered at the same time (for right-to-left layouts).

StyleBox **normal**

Default StyleBox for the menu item.

StyleBox **normal_mirrored** 🔗

Default StyleBox for the menu item (for right-to-left layouts).

StyleBox **pressed** 🔗

StyleBox used when the menu item is being pressed.

StyleBox **pressed_mirrored** 🔗

StyleBox used when the menu item is being pressed (for right-to-left layouts).

MenuButton

Inherits: Button **<** BaseButton **<** Control **<** CanvasItem **<** Node **<** Object

A button that brings up a PopupMenu when clicked.

Description

A button that brings up a PopupMenu when clicked. To create new items inside this PopupMenu, use `get_popup().add_item("My Item Name")`. You can also create them directly from Godot editor's inspector.

See also BaseButton which contains common properties and methods associated with this node.

Properties

ActionMode	action_mode	`0` (overrides BaseButton)
bool	flat	`true` (overrides Button)
FocusMode	focus_mode	`0` (overrides Control)
int	item_count	`0`

bool	switch_on_hover	`false`
bool	toggle_mode	`true` (overrides BaseButton)

Methods

PopupMenu	get_popup() const
void	set_disable_shortcuts(disabled: bool)
void	show_popup()

Signals

about_to_popup() 🔗

Emitted when the PopupMenu of this MenuButton is about to show.

Property Descriptions

int **item_count** = `0` 🔗

- void **set_item_count**(value: int)
- int **get_item_count**()

The number of items currently in the list.

---

bool **switch_on_hover** = `false` 🔗

- void **set_switch_on_hover**(value: bool)
- bool **is_switch_on_hover**()

If `true`, when the cursor hovers above another **MenuButton** within the same parent which also has switch_on_hover enabled, it will close the current **MenuButton** and open the other one.

Method Descriptions

PopupMenu **get_popup**() const 🔗

Returns the PopupMenu contained in this button.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their Window.visible property.

---

void **set_disable_shortcuts**(disabled: bool) 🔗

If `true`, shortcuts are disabled and cannot be used to trigger the button.

---

void **show_popup**() 🔗

Adjusts popup position and sizing for the **MenuButton**, then shows the PopupMenu. Prefer this over using `get_popup().popup()`.

MeshInstance2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Node used for displaying a Mesh in 2D.

Description

Node used for displaying a Mesh in 2D. A **MeshInstance2D** can be automatically created from an existing Sprite2D via a tool in the editor toolbar. Select the Sprite2D node, then choose **Sprite2D > Convert to MeshInstance2D** at the top of the 2D editor viewport.

Tutorials

- 2D meshes

Properties

Mesh	mesh
Texture2D	texture

Signals

texture_changed() 🔗

Emitted when the texture is changed.

Property Descriptions

Mesh **mesh** 🔗

- void **set_mesh**(value: Mesh)
- Mesh **get_mesh**()

The Mesh that will be drawn by the **MeshInstance2D**.

Texture2D **texture** 🔗

- void **set_texture**(value: Texture2D)
- Texture2D **get_texture**()

The Texture2D that will be used if using the default CanvasItemMaterial. Can be accessed as `TEXTURE` in CanvasItem shader.

MeshInstance3D

Inherits: GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

Inherited By: SoftBody3D

Node that instances meshes into a scenario.

Description

MeshInstance3D is a node that takes a Mesh resource and adds it to the current scenario by creating an instance of it. This is the class most often used render 3D geometry and can be used to instance a single Mesh in many places. This allows reusing geometry, which can save on resources. When a Mesh has to be instantiated more than thousands of times at close proximity, consider using a MultiMesh in a MultiMeshInstance3D instead.

Tutorials

- 3D Material Testers Demo [https://godotengine.org/asset-library/asset/2742]
- 3D Kinematic Character Demo [https://godotengine.org/asset-library/asset/2739]
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

Mesh	mesh	
NodePath	skeleton	`NodePath("..")`
Skin	skin	

Methods

ArrayMesh	bake_mesh_from_current_blend_shape_mix(existing: ArrayMesh = null)
void	create_convex_collision(clean: bool = true, simplify: bool = false)
void	create_debug_tangents()
void	create_multiple_convex_collisions(settings: MeshConvexDecompositionSettings = null)
void	create_trimesh_collision()
int	find_blend_shape_by_name(name: StringName)
Material	get_active_material(surface: int) const

int	get_blend_shape_count() const
float	get_blend_shape_value(blend_shape_idx: int) const
SkinReference	get_skin_reference() const
Material	get_surface_override_material(surface: int) const
int	get_surface_override_material_count() const
void	set_blend_shape_value(blend_shape_idx: int, value: float)
void	set_surface_override_material(surface: int, material: Material)

Property Descriptions

Mesh **mesh** 🔗

- void **set_mesh**(value: Mesh)
- Mesh **get_mesh**()

The Mesh resource for the instance.

NodePath **skeleton** = `NodePath("..")` 🔗

- void **set_skeleton_path**(value: NodePath)
- NodePath **get_skeleton_path**()

NodePath to the Skeleton3D associated with the instance.

Skin **skin**

- void **set_skin**(value: Skin)
- Skin **get_skin**()

The Skin to be used by this instance.

Method Descriptions

ArrayMesh **bake_mesh_from_current_blend_shape_mix**(existing: ArrayMesh = null)

Takes a snapshot from the current ArrayMesh with all blend shapes applied according to their current weights and bakes it to the provided `existing` mesh. If no `existing` mesh is provided a new ArrayMesh is created, baked and returned. Mesh surface materials are not copied.

Performance: Mesh data needs to be received from the GPU, stalling the RenderingServer in the process.

void **create_convex_collision**(clean: bool = true, simplify: bool = false)

This helper creates a StaticBody3D child node with a ConvexPolygonShape3D collision shape calculated from the mesh geometry. It's mainly used for testing.

If `clean` is `true` (default), duplicate and interior vertices are removed automatically. You can set it to `false` to make the process faster if not needed.

If `simplify` is `true`, the geometry can be further simplified to reduce the number of vertices. Disabled by default.

void **create_debug_tangents**() 🔗

This helper creates a **MeshInstance3D** child node with gizmos at every vertex calculated from the mesh geometry. It's mainly used for testing.

void **create_multiple_convex_collisions**(settings: MeshConvexDecompositionSettings = null) 🔗

This helper creates a StaticBody3D child node with multiple ConvexPolygonShape3D collision shapes calculated from the mesh geometry via convex decomposition. The convex decomposition operation can be controlled with parameters from the optional `settings`.

void **create_trimesh_collision**() 🔗

This helper creates a StaticBody3D child node with a ConcavePolygonShape3D collision shape calculated from the mesh geometry. It's mainly used for testing.

int **find_blend_shape_by_name**(name: StringName) 🔗

Returns the index of the blend shape with the given `name`. Returns `-1` if no blend shape with this name exists, including when mesh is `null`.

Material **get_active_material**(surface: int) const 🔗

Returns the Material that will be used by the Mesh when drawing. This can return the GeometryInstance3D.material_override, the surface override Material defined in this **MeshInstance3D**, or the surface Material defined in the mesh. For example, if GeometryInstance3D.material_override is used, all surfaces will return the override material.

Returns `null` if no material is active, including when mesh is `null`.

---

int **get_blend_shape_count**() const 🔗

Returns the number of blend shapes available. Produces an error if mesh is `null`.

---

float **get_blend_shape_value**(blend_shape_idx: int) const 🔗

Returns the value of the blend shape at the given `blend_shape_idx`. Returns `0.0` and produces an error if mesh is `null` or doesn't have a blend shape at that index.

SkinReference **get_skin_reference**() const 🔗

Returns the internal SkinReference containing the skeleton's RID attached to this RID. See also Resource.get_rid, SkinReference.get_skeleton, and RenderingServer.instance_attach_skeleton.

Material **get_surface_override_material**(surface: int) const 🔗

Returns the override Material for the specified `surface` of the Mesh resource. See also get_surface_override_material_count.

Note: This returns the Material associated to the **MeshInstance3D**'s Surface Material Override properties, not the material within the Mesh resource. To get the material within the Mesh resource, use Mesh.surface_get_material instead.

int **get_surface_override_material_count**() const 🔗

Returns the number of surface override materials. This is equivalent to Mesh.get_surface_count. See also get_surface_override_material.

void **set_blend_shape_value**(blend_shape_idx: int, value: float) 🔗

Sets the value of the blend shape at `blend_shape_idx` to `value`. Produces an error if mesh is `null` or doesn't have a blend shape at that index.

void **set_surface_override_material**(surface: int, material: Material) 🔗

Sets the override `material` for the specified `surface` of the Mesh resource. This material is associated with this **MeshInstance3D** rather than with mesh.

Note: This assigns the Material associated to the **MeshInstance3D**'s Surface Material Override properties, not the material within the Mesh resource. To set the material within the Mesh resource, use Mesh.surface_get_material instead.

MissingNode

Inherits: Node **<** Object

An internal editor class intended for keeping the data of unrecognized nodes.

Description

This is an internal editor class intended for keeping data of nodes of unknown type (most likely this type was supplied by an extension that is no longer loaded). It can't be manually instantiated or placed in a scene.

Warning: Ignore missing nodes unless you know what you are doing. Existing properties on a missing node can be freely modified in code, regardless of the type they are intended to be.

Properties

String	original_class
String	original_scene
bool	recording_properties

Property Descriptions

String **original_class** 🔗

- void **set_original_class**(value: String)
- String **get_original_class**()

The name of the class this node was supposed to be (see Object.get_class).

String **original_scene** 🔗

- void **set_original_scene**(value: String)
- String **get_original_scene**()

Returns the path of the scene this node was instance of originally.

---

bool **recording_properties** 🔗

- void **set_recording_properties**(value: bool)
- bool **is_recording_properties**()

If `true`, allows new properties to be set along with existing ones. If `false`, only existing properties' values can be set, and new properties cannot be added.

MultiMeshInstance2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Node that instances a MultiMesh in 2D.

Description

MultiMeshInstance2D is a specialized node to instance a MultiMesh resource in 2D.

Usage is the same as MultiMeshInstance3D.

Properties

MultiMesh	multimesh
Texture2D	texture

Signals

texture_changed() 🔗

Emitted when the texture is changed.

Property Descriptions

MultiMesh **multimesh** 🔗

- void **set_multimesh**(value: MultiMesh)
- MultiMesh **get_multimesh**()

The MultiMesh that will be drawn by the **MultiMeshInstance2D**.

Texture2D **texture** 🔗

- void **set_texture**(value: Texture2D)
- Texture2D **get_texture**()

The Texture2D that will be used if using the default CanvasItemMaterial. Can be accessed as `TEXTURE` in CanvasItem shader.

MultiMeshInstance3D

Inherits: GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

Node that instances a MultiMesh.

Description

MultiMeshInstance3D is a specialized node to instance GeometryInstance3Ds based on a MultiMesh resource.

This is useful to optimize the rendering of a high number of instances of a given mesh (for example trees in a forest or grass strands).

Tutorials

- Using MultiMeshInstance
- Optimization using MultiMeshes
- Animating thousands of fish with MultiMeshInstance

Properties

MultiMesh	multimesh

Property Descriptions

MultiMesh **multimesh**

- void **set_multimesh**(value: MultiMesh)
- MultiMesh **get_multimesh**()

The MultiMesh resource that will be used and shared among all instances of the **MultiMeshInstance3D**.

MultiplayerSpawner

Inherits: Node **<** Object

Automatically replicates spawnable nodes from the authority to other multiplayer peers.

Description

Spawnable scenes can be configured in the editor or through code (see add_spawnable_scene).

Also supports custom node spawns through spawn, calling spawn_function on all peers.

Internally, **MultiplayerSpawner** uses MultiplayerAPI.object_configuration_add to notify spawns passing the spawned node as the `object` and itself as the `configuration`, and MultiplayerAPI.object_configuration_remove to notify despawns in a similar way.

Properties

Callable	spawn_function	
int	spawn_limit	`0`
NodePath	spawn_path	`NodePath("")`

Methods

void	add_spawnable_scene(path: String)
void	clear_spawnable_scenes()
String	get_spawnable_scene(index: int) const
int	get_spawnable_scene_count() const
Node	spawn(data: Variant = null)

Signals

despawned(node: Node)

Emitted when a spawnable scene or custom spawn was despawned by the multiplayer authority. Only called on puppets.

spawned(node: Node)

Emitted when a spawnable scene or custom spawn was spawned by the multiplayer authority. Only called on puppets.

Property Descriptions

Callable **spawn_function**

- void **set_spawn_function**(value: Callable)
- Callable **get_spawn_function**()

Method called on all peers when a custom spawn is requested by the authority. Will receive the `data` parameter, and should return a Node that is not in the scene tree.

Note: The returned node should **not** be added to the scene with Node.add_child. This is done automatically.

---

int **spawn_limit** = `0`

- void **set_spawn_limit**(value: int)
- int **get_spawn_limit**()

Maximum number of nodes allowed to be spawned by this spawner. Includes both spawnable scenes and custom spawns.

When set to `0` (the default), there is no limit.

NodePath **spawn_path** = `NodePath("")`

- void **set_spawn_path**(value: NodePath)
- NodePath **get_spawn_path**()

Path to the spawn root. Spawnable scenes that are added as direct children are replicated to other peers.

Method Descriptions

void **add_spawnable_scene**(path: String)

Adds a scene path to spawnable scenes, making it automatically replicated from the multiplayer authority to other peers when added as children of the node pointed by spawn_path.

---

void **clear_spawnable_scenes**()

Clears all spawnable scenes. Does not despawn existing instances on remote peers.

String **get_spawnable_scene**(index: int) const

Returns the spawnable scene path by index.

---

int **get_spawnable_scene_count**() const

Returns the count of spawnable scene paths.

Node **spawn**(data: Variant = null)

Requests a custom spawn, with `data` passed to spawn_function on all peers. Returns the locally spawned node instance already inside the scene tree, and added as a child of the node pointed by spawn_path.

Note: Spawnable scenes are spawned automatically. spawn is only needed for custom spawns.

MultiplayerSynchronizer

Inherits: Node < Object

Synchronizes properties from the multiplayer authority to the remote peers.

Description

By default, **MultiplayerSynchronizer** synchronizes configured properties to all peers.

Visibility can be handled directly with set_visibility_for or as-needed with add_visibility_filter and update_visibility.

MultiplayerSpawners will handle nodes according to visibility of synchronizers as long as the node at root_path was spawned by one.

Internally, **MultiplayerSynchronizer** uses MultiplayerAPI.object_configuration_add to notify synchronization start passing the Node at root_path as the `object` and itself as the `configuration`, and uses MultiplayerAPI.object_configuration_remove to notify synchronization end in a similar way.

Note: Synchronization is not supported for Object type properties, like Resource. Properties that are unique to each peer, like the instance IDs of Objects (see Object.get_instance_id) or RIDs, will also not work in synchronization.

Properties

float	delta_interval	`0.0`
bool	public_visibility	`true`
SceneReplicationConfig	replication_config	
float	replication_interval	`0.0`
NodePath	root_path	`NodePath("..")`
VisibilityUpdateMode	visibility_update_mode	`0`

Methods

void	add_visibility_filter(filter: Callable)
bool	get_visibility_for(peer: int) const
void	remove_visibility_filter(filter: Callable)
void	set_visibility_for(peer: int, visible: bool)
void	update_visibility(for_peer: int = 0)

Signals

delta_synchronized() 🔗

Emitted when a new delta synchronization state is received by this synchronizer after the properties have been updated.

---

synchronized() 🔗

Emitted when a new synchronization state is received by this synchronizer after the properties have been updated.

---

visibility_changed(for_peer: int) 🔗

Emitted when visibility of `for_peer` is updated. See update_visibility.

Enumerations

enum **VisibilityUpdateMode**: 🔗

VisibilityUpdateMode **VISIBILITY_PROCESS_IDLE** = `0`

Visibility filters are updated during process frames (see Node.NOTIFICATION_INTERNAL_PROCESS).

VisibilityUpdateMode **VISIBILITY_PROCESS_PHYSICS** = `1`

Visibility filters are updated during physics frames (see Node.NOTIFICATION_INTERNAL_PHYSICS_PROCESS).

VisibilityUpdateMode **VISIBILITY_PROCESS_NONE** = `2`

Visibility filters are not updated automatically, and must be updated manually by calling update_visibility.

Property Descriptions

float **delta_interval** = `0.0` 🔗

- void **set_delta_interval**(value: float)
- float **get_delta_interval**()

Time interval between delta synchronizations. When set to `0.0` (the default), delta synchronizations happen every network process frame.

---

bool **public_visibility** = `true` 🔗

- void **set_visibility_public**(value: bool)
- bool **is_visibility_public**()

Whether synchronization should be visible to all peers by default. See set_visibility_for and add_visibility_filter for ways of configuring fine-grained visibility options.

SceneReplicationConfig **replication_config** 🔗

- void **set_replication_config**(value: SceneReplicationConfig)
- SceneReplicationConfig **get_replication_config**()

Resource containing which properties to synchronize.

---

float **replication_interval** = `0.0` 🔗

- void **set_replication_interval**(value: float)
- float **get_replication_interval**()

Time interval between synchronizations. When set to `0.0` (the default), synchronizations happen every network process frame.

NodePath **root_path** = `NodePath("..")`

- void **set_root_path**(value: NodePath)
- NodePath **get_root_path**()

Node path that replicated properties are relative to.

If root_path was spawned by a MultiplayerSpawner, the node will be also be spawned and despawned based on this synchronizer visibility options.

VisibilityUpdateMode **visibility_update_mode** = `0`

- void **set_visibility_update_mode**(value: VisibilityUpdateMode)
- VisibilityUpdateMode **get_visibility_update_mode**()

Specifies when visibility filters are updated (see VisibilityUpdateMode for options).

Method Descriptions

void **add_visibility_filter**(filter: Callable)

Adds a peer visibility filter for this synchronizer.

`filter` should take a peer ID int and return a bool.

---

bool **get_visibility_for**(peer: int) const 🔗

Queries the current visibility for peer `peer`.

---

void **remove_visibility_filter**(filter: Callable) 🔗

Removes a peer visibility filter from this synchronizer.

---

void **set_visibility_for**(peer: int, visible: bool) 🔗

Sets the visibility of `peer` to `visible`. If `peer` is `0`, the value of public_visibility will be updated instead.

---

void **update_visibility**(for_peer: int = 0) 🔗

Updates the visibility of `for_peer` according to visibility filters. If `for_peer` is `0` (the default), all peers' visibilties are updated.

NavigationAgent2D

Experimental: This class may be changed or removed in future versions.

Inherits: Node **<** Object

A 2D agent used to pathfind to a position while avoiding obstacles.

Description

A 2D agent used to pathfind to a position while avoiding static and dynamic obstacles. The calculation can be used by the parent node to dynamically move it along the path. Requires navigation data to work correctly.

Dynamic obstacles are avoided using RVO collision avoidance. Avoidance is computed before physics, so the pathfinding information can be used safely in the physics step.

Note: After setting the target_position property, the get_next_path_position method must be used once every physics frame to update the internal path logic of the navigation agent. The vector position it returns should be used as the next movement position for the agent's parent node.

Tutorials

- Using NavigationAgents

Properties

bool	avoidance_enabled	`false`

int	avoidance_layers	`1`
int	avoidance_mask	`1`
float	avoidance_priority	`1.0`
bool	debug_enabled	`false`
Color	debug_path_custom_color	`Color(1, 1, 1, 1)`
float	debug_path_custom_line_width	`-1.0`
float	debug_path_custom_point_size	`4.0`
bool	debug_use_custom	`false`
int	max_neighbors	`10`
float	max_speed	`100.0`
int	navigation_layers	`1`
float	neighbor_distance	`500.0`
float	path_desired_distance	`20.0`

float	path_max_distance	`100.0`
BitField[PathMetadataFlags]	path_metadata_flags	`7`
PathPostProcessing	path_postprocessing	`0`
PathfindingAlgorithm	pathfinding_algorithm	`0`
float	radius	`10.0`
float	simplify_epsilon	`0.0`
bool	simplify_path	`false`
float	target_desired_distance	`10.0`
Vector2	target_position	`Vector2(0, 0)`
float	time_horizon_agents	`1.0`
float	time_horizon_obstacles	`0.0`
Vector2	velocity	`Vector2(0, 0)`

Methods

float	distance_to_target() const
bool	get_avoidance_layer_value(layer_number: int) const
bool	get_avoidance_mask_value(mask_number: int) const
PackedVector2Array	get_current_navigation_path() const
int	get_current_navigation_path_index() const
NavigationPathQueryResult2D	get_current_navigation_result() const
Vector2	get_final_position()
bool	get_navigation_layer_value(layer_number: int) const
RID	get_navigation_map() const
Vector2	get_next_path_position()
RID	get_rid() const
bool	is_navigation_finished()
bool	is_target_reachable()

bool	is_target_reached() const
void	set_avoidance_layer_value(layer_number: int, value: bool)
void	set_avoidance_mask_value(mask_number: int, value: bool)
void	set_navigation_layer_value(layer_number: int, value: bool)
void	set_navigation_map(navigation_map: RID)
void	set_velocity_forced(velocity: Vector2)

Signals

link_reached(details: Dictionary) 🔗

Signals that the agent reached a navigation link. Emitted when the agent moves within path_desired_distance of the next position of the path when that position is a navigation link.

The details dictionary may contain the following keys depending on the value of path_metadata_flags:

- `position`: The start position of the link that was reached.
- `type`: Always NavigationPathQueryResult2D.PATH_SEGMENT_TYPE_LINK.
- `rid`: The RID of the link.
- `owner`: The object which manages the link (usually NavigationLink2D).

- `link_entry_position`: If `owner` is available and the owner is a NavigationLink2D, it will contain the global position of the link's point the agent is entering.
- `link_exit_position`: If `owner` is available and the owner is a NavigationLink2D, it will contain the global position of the link's point which the agent is exiting.

navigation_finished() 🔗

Signals that the agent's navigation has finished. If the target is reachable, navigation ends when the target is reached. If the target is unreachable, navigation ends when the last waypoint of the path is reached. This signal is emitted only once per loaded path.

This signal will be emitted just after target_reached when the target is reachable.

path_changed() 🔗

Emitted when the agent had to update the loaded path:

- because path was previously empty.
- because navigation map has changed.
- because agent pushed further away from the current path segment than the path_max_distance.

target_reached() 🔗

Signals that the agent reached the target, i.e. the agent moved within target_desired_distance of the target_position. This signal is emitted only once per loaded path.

This signal will be emitted just before navigation_finished when the target is reachable.

It may not always be possible to reach the target but it should always be possible to reach the final position. See get_final_position.

---

velocity_computed(safe_velocity: Vector2)

Notifies when the collision avoidance velocity is calculated. Emitted every update as long as avoidance_enabled is `true` and the agent has a navigation map.

---

waypoint_reached(details: Dictionary)

Signals that the agent reached a waypoint. Emitted when the agent moves within path_desired_distance of the next position of the path.

The details dictionary may contain the following keys depending on the value of path_metadata_flags:

- `position`: The position of the waypoint that was reached.
- `type`: The type of navigation primitive (region or link) that contains this waypoint.
- `rid`: The RID of the containing navigation primitive (region or link).
- `owner`: The object which manages the containing navigation primitive (region or link).

Property Descriptions

bool **avoidance_enabled** = `false`

- void **set_avoidance_enabled**(value: bool)
- bool **get_avoidance_enabled**()

If `true` the agent is registered for an RVO avoidance callback on the NavigationServer2D. When velocity is used and the processing is completed a `safe_velocity` Vector2 is received with a signal connection to velocity_computed. Avoidance processing with many registered agents has a significant performance cost and should only be enabled on agents that currently require it.

---

int **avoidance_layers** = `1`

- void **set_avoidance_layers**(value: int)
- int **get_avoidance_layers**()

A bitfield determining the avoidance layers for this NavigationAgent. Other agents with a matching bit on the avoidance_mask will avoid this agent.

---

int **avoidance_mask** = `1`

- void **set_avoidance_mask**(value: int)
- int **get_avoidance_mask**()

A bitfield determining what other avoidance agents and obstacles this NavigationAgent will avoid when a bit matches at least one of their avoidance_layers.

---

float **avoidance_priority** = `1.0`

- void **set_avoidance_priority**(value: float)
- float **get_avoidance_priority**()

The agent does not adjust the velocity for other agents that would match the avoidance_mask but have a lower avoidance_priority. This in turn makes the other agents with lower priority adjust their velocities even more to avoid collision with this agent.

---

bool **debug_enabled** = `false`

- void **set_debug_enabled**(value: bool)
- bool **get_debug_enabled**()

If `true` shows debug visuals for this agent.

Color **debug_path_custom_color** = `Color(1, 1, 1, 1)`

- void **set_debug_path_custom_color**(value: Color)
- Color **get_debug_path_custom_color**()

If debug_use_custom is `true` uses this color for this agent instead of global color.

---

float **debug_path_custom_line_width** = `-1.0`

- void **set_debug_path_custom_line_width**(value: float)
- float **get_debug_path_custom_line_width**()

If debug_use_custom is `true` uses this line width for rendering paths for this agent instead of global line width.

---

float **debug_path_custom_point_size** = `4.0`

- void **set_debug_path_custom_point_size**(value: float)
- float **get_debug_path_custom_point_size**()

If debug_use_custom is `true` uses this rasterized point size for rendering path points for this agent instead of global point size.

---

bool **debug_use_custom** = `false`

- void **set_debug_use_custom**(value: bool)
- bool **get_debug_use_custom**()

If `true` uses the defined debug_path_custom_color for this agent instead of global color.

---

int **max_neighbors** = `10`

- void **set_max_neighbors**(value: int)
- int **get_max_neighbors**()

The maximum number of neighbors for the agent to consider.

---

float **max_speed** = `100.0`

- void **set_max_speed**(value: float)
- float **get_max_speed**()

The maximum speed that an agent can move.

---

int **navigation_layers** = `1`

- void **set_navigation_layers**(value: int)
- int **get_navigation_layers**()

A bitfield determining which navigation layers of navigation regions this agent will use to calculate a path. Changing it during runtime will clear the current navigation path and generate a new one, according to the new navigation layers.

---

float **neighbor_distance** = `500.0`

- void **set_neighbor_distance**(value: float)
- float **get_neighbor_distance**()

The distance to search for other agents.

---

float **path_desired_distance** = `20.0`

- void **set_path_desired_distance**(value: float)
- float **get_path_desired_distance**()

The distance threshold before a path point is considered to be reached. This allows agents to not have to hit a path point on the path exactly, but only to reach its general area. If this value is set too high, the NavigationAgent will skip points on the path, which can lead to it leaving the navigation mesh. If this value is set too low, the NavigationAgent will be stuck in a repath loop because it will constantly overshoot the distance to the next point on each physics frame update.

---

float **path_max_distance** = `100.0` 🔗

- void **set_path_max_distance**(value: float)
- float **get_path_max_distance**()

The maximum distance the agent is allowed away from the ideal path to the final position. This can happen due to trying to avoid collisions. When the maximum distance is exceeded, it recalculates the ideal path.

BitField[PathMetadataFlags] **path_metadata_flags** = `7` 🔗

- void **set_path_metadata_flags**(value: BitField[PathMetadataFlags])
- BitField[PathMetadataFlags] **get_path_metadata_flags**()

Additional information to return with the navigation path.

PathPostProcessing **path_postprocessing** = `0` 🔗

- void **set_path_postprocessing**(value: PathPostProcessing)
- PathPostProcessing **get_path_postprocessing**()

The path postprocessing applied to the raw path corridor found by the pathfinding_algorithm.

PathfindingAlgorithm **pathfinding_algorithm** = `0` 🔗

- void **set_pathfinding_algorithm**(value: PathfindingAlgorithm)
- PathfindingAlgorithm **get_pathfinding_algorithm**()

The pathfinding algorithm used in the path query.

---

float **radius** = `10.0` 🔗

- void **set_radius**(value: float)
- float **get_radius**()

The radius of the avoidance agent. This is the "body" of the avoidance agent and not the avoidance maneuver starting radius (which is controlled by neighbor_distance).

Does not affect normal pathfinding. To change an actor's pathfinding radius bake NavigationMesh resources with a different NavigationMesh.agent_radius property and use different navigation maps for each actor size.

---

float **simplify_epsilon** = `0.0`

- void **set_simplify_epsilon**(value: float)
- float **get_simplify_epsilon**()

The path simplification amount in worlds units.

---

bool **simplify_path** = `false`

- void **set_simplify_path**(value: bool)
- bool **get_simplify_path**()

If `true` a simplified version of the path will be returned with less critical path points removed. The simplification amount is controlled by simplify_epsilon. The simplification uses a variant of Ramer-Douglas-Peucker algorithm for curve point decimation.

Path simplification can be helpful to mitigate various path following issues that can arise with certain agent types and script behaviors. E.g. "steering" agents or avoidance in "open fields".

---

float **target_desired_distance** = `10.0`

- void **set_target_desired_distance**(value: float)
- float **get_target_desired_distance**()

The distance threshold before the target is considered to be reached. On reaching the target, target_reached is emitted and navigation ends (see is_navigation_finished and navigation_finished).

You can make navigation end early by setting this property to a value greater than path_desired_distance (navigation will end before reaching the last waypoint).

You can also make navigation end closer to the target than each individual path position by setting this property to a value lower than path_desired_distance (navigation won't immediately end when reaching the last waypoint). However, if the value set is too low, the agent will be stuck in a repath loop because it will constantly overshoot the distance to the target on each physics frame update.

Vector2 **target_position** = `Vector2(0, 0)`

- void **set_target_position**(value: Vector2)
- Vector2 **get_target_position**()

If set, a new navigation path from the current agent position to the target_position is requested from the NavigationServer.

---

float **time_horizon_agents** = `1.0`

- void **set_time_horizon_agents**(value: float)
- float **get_time_horizon_agents**()

The minimal amount of time for which this agent's velocities, that are computed with the collision avoidance algorithm, are safe with respect to other agents. The larger the number, the sooner the agent will respond to other agents, but less freedom in choosing its velocities. A too high value will slow down agents movement considerably. Must be positive.

---

float **time_horizon_obstacles** = `0.0`

- void **set_time_horizon_obstacles**(value: float)
- float **get_time_horizon_obstacles**()

The minimal amount of time for which this agent's velocities, that are computed with the collision avoidance algorithm, are safe with respect to static avoidance obstacles. The larger the number, the sooner the agent will respond to static avoidance obstacles, but less freedom in choosing its velocities. A too high value will slow down agents movement considerably. Must be positive.

Vector2 **velocity** = `Vector2(0, 0)`

- void **set_velocity**(value: Vector2)
- Vector2 **get_velocity**()

Sets the new wanted velocity for the agent. The avoidance simulation will try to fulfill this velocity if possible but will modify it to avoid collision with other agents and obstacles. When an agent is teleported to a new position, use set_velocity_forced as well to reset the internal simulation velocity.

Method Descriptions

float **distance_to_target**() const

Returns the distance to the target position, using the agent's global position. The user must set target_position in order for this to be accurate.

---

bool **get_avoidance_layer_value**(layer_number: int) const

Returns whether or not the specified layer of the avoidance_layers bitmask is enabled, given a `layer_number` between 1 and 32.

---

bool **get_avoidance_mask_value**(mask_number: int) const

Returns whether or not the specified mask of the avoidance_mask bitmask is enabled, given a `mask_number` between 1 and 32.

PackedVector2Array **get_current_navigation_path**() const

Returns this agent's current path from start to finish in global coordinates. The path only updates when the target position is changed or the agent requires a repath. The path array is not intended to be used in direct path movement as the agent has its own internal path logic that would get corrupted by changing the path array manually. Use the intended get_next_path_position once every physics frame to receive the next path point for the agents movement as this function also updates the internal path logic.

---

int **get_current_navigation_path_index**() const

Returns which index the agent is currently on in the navigation path's PackedVector2Array.

NavigationPathQueryResult2D **get_current_navigation_result**() const

Returns the path query result for the path the agent is currently following.

Vector2 **get_final_position**()

Returns the reachable final position of the current navigation path in global coordinates. This position can change if the agent needs to update the navigation path which makes the agent emit the path_changed signal.

---

bool **get_navigation_layer_value**(layer_number: int) const

Returns whether or not the specified layer of the navigation_layers bitmask is enabled, given a `layer_number` between 1 and 32.

RID **get_navigation_map**() const

Returns the RID of the navigation map for this NavigationAgent node. This function returns always the map set on the NavigationAgent node and not the map of the abstract agent on the NavigationServer. If the agent map is changed directly with the NavigationServer API the NavigationAgent node will not be aware of the map change. Use set_navigation_map to change the navigation map for the NavigationAgent and also update the agent on the NavigationServer.

Vector2 **get_next_path_position**() 🔗

Returns the next position in global coordinates that can be moved to, making sure that there are no static objects in the way. If the agent does not have a navigation path, it will return the position of the agent's parent. The use of this function once every physics frame is required to update the internal path logic of the NavigationAgent.

RID **get_rid**() const 🔗

Returns the RID of this agent on the NavigationServer2D.

---

bool **is_navigation_finished**() 🔗

Returns `true` if the agent's navigation has finished. If the target is reachable, navigation ends when the target is reached. If the target is unreachable, navigation ends when the last waypoint of the path is reached.

Note: While `true` prefer to stop calling update functions like get_next_path_position. This avoids jittering the standing agent due to calling repeated path updates.

---

bool **is_target_reachable**() 🔗

Returns `true` if get_final_position is within target_desired_distance of the target_position.

---

bool **is_target_reached**() const

Returns `true` if the agent reached the target, i.e. the agent moved within target_desired_distance of the target_position. It may not always be possible to reach the target but it should always be possible to reach the final position. See get_final_position.

---

void **set_avoidance_layer_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the avoidance_layers bitmask, given a `layer_number` between 1 and 32.

---

void **set_avoidance_mask_value**(mask_number: int, value: bool)

Based on `value`, enables or disables the specified mask in the avoidance_mask bitmask, given a `mask_number` between 1 and 32.

---

void **set_navigation_layer_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the navigation_layers bitmask, given a `layer_number` between 1 and 32.

---

void **set_navigation_map**(navigation_map: RID)

Sets the RID of the navigation map this NavigationAgent node should use and also updates the `agent` on the NavigationServer.

---

void **set_velocity_forced**(velocity: Vector2)

Replaces the internal velocity in the collision avoidance simulation with `velocity`. When an agent is teleported to a new position this function should be used in the same frame. If called frequently this function can get agents stuck.

NavigationAgent3D

Experimental: This class may be changed or removed in future versions.

Inherits: Node **<** Object

A 3D agent used to pathfind to a position while avoiding obstacles.

Description

A 3D agent used to pathfind to a position while avoiding static and dynamic obstacles. The calculation can be used by the parent node to dynamically move it along the path. Requires navigation data to work correctly.

Dynamic obstacles are avoided using RVO collision avoidance. Avoidance is computed before physics, so the pathfinding information can be used safely in the physics step.

Note: After setting the target_position property, the get_next_path_position method must be used once every physics frame to update the internal path logic of the navigation agent. The vector position it returns should be used as the next movement position for the agent's parent node.

Tutorials

- Using NavigationAgents

Properties

bool	avoidance_enabled	`false`

int	avoidance_layers	`1`
int	avoidance_mask	`1`
float	avoidance_priority	`1.0`
bool	debug_enabled	`false`
Color	debug_path_custom_color	`Color(1, 1, 1, 1)`
float	debug_path_custom_point_size	`4.0`
bool	debug_use_custom	`false`
float	height	`1.0`
bool	keep_y_velocity	`true`
int	max_neighbors	`10`
float	max_speed	`10.0`
int	navigation_layers	`1`
float	neighbor_distance	`50.0`

float	path_desired_distance	`1.0`
float	path_height_offset	`0.0`
float	path_max_distance	`5.0`
BitField[PathMetadataFlags]	path_metadata_flags	`7`
PathPostProcessing	path_postprocessing	`0`
PathfindingAlgorithm	pathfinding_algorithm	`0`
float	radius	`0.5`
float	simplify_epsilon	`0.0`
bool	simplify_path	`false`
float	target_desired_distance	`1.0`
Vector3	target_position	`Vector3(0, 0, 0)`
float	time_horizon_agents	`1.0`
float	time_horizon_obstacles	`0.0`

bool	use_3d_avoidance	`false`
Vector3	velocity	`Vector3(0, 0, 0)`

Methods

float	distance_to_target() const
bool	get_avoidance_layer_value(layer_number: int) const
bool	get_avoidance_mask_value(mask_number: int) const
PackedVector3Array	get_current_navigation_path() const
int	get_current_navigation_path_index() const
NavigationPathQueryResult3D	get_current_navigation_result() const
Vector3	get_final_position()
bool	get_navigation_layer_value(layer_number: int) const
RID	get_navigation_map() const

Vector3	get_next_path_position()
RID	get_rid() const
bool	is_navigation_finished()
bool	is_target_reachable()
bool	is_target_reached() const
void	set_avoidance_layer_value(layer_number: int, value: bool)
void	set_avoidance_mask_value(mask_number: int, value: bool)
void	set_navigation_layer_value(layer_number: int, value: bool)
void	set_navigation_map(navigation_map: RID)
void	set_velocity_forced(velocity: Vector3)

Signals

link_reached(details: Dictionary) 🔗

Signals that the agent reached a navigation link. Emitted when the agent moves within path_desired_distance of the next position of the path when that position is a navigation link.

The details dictionary may contain the following keys depending on the value of path_metadata_flags:

- `position`: The start position of the link that was reached.
- `type`: Always NavigationPathQueryResult3D.PATH_SEGMENT_TYPE_LINK.
- `rid`: The RID of the link.
- `owner`: The object which manages the link (usually NavigationLink3D).
- `link_entry_position`: If `owner` is available and the owner is a NavigationLink3D, it will contain the global position of the link's point the agent is entering.
- `link_exit_position`: If `owner` is available and the owner is a NavigationLink3D, it will contain the global position of the link's point which the agent is exiting.

---

navigation_finished()

Signals that the agent's navigation has finished. If the target is reachable, navigation ends when the target is reached. If the target is unreachable, navigation ends when the last waypoint of the path is reached. This signal is emitted only once per loaded path.

This signal will be emitted just after target_reached when the target is reachable.

---

path_changed()

Emitted when the agent had to update the loaded path:

- because path was previously empty.
- because navigation map has changed.
- because agent pushed further away from the current path segment than the path_max_distance.

---

target_reached()

Signals that the agent reached the target, i.e. the agent moved within target_desired_distance of the target_position. This signal is emitted only once per loaded path.

This signal will be emitted just before navigation_finished when the target is reachable.

It may not always be possible to reach the target but it should always be possible to reach the final position. See get_final_position.

---

velocity_computed(safe_velocity: Vector3)

Notifies when the collision avoidance velocity is calculated. Emitted every update as long as avoidance_enabled is `true` and the agent has a navigation map.

---

waypoint_reached(details: Dictionary)

Signals that the agent reached a waypoint. Emitted when the agent moves within path_desired_distance of the next position of the path.

The details dictionary may contain the following keys depending on the value of path_metadata_flags:

- `position`: The position of the waypoint that was reached.
- `type`: The type of navigation primitive (region or link) that contains this waypoint.
- `rid`: The RID of the containing navigation primitive (region or link).
- `owner`: The object which manages the containing navigation primitive (region or link).

Property Descriptions

bool **avoidance_enabled** = `false` 🔗

- void **set_avoidance_enabled**(value: bool)
- bool **get_avoidance_enabled**()

If `true` the agent is registered for an RVO avoidance callback on the NavigationServer3D. When velocity is set and the processing is completed a `safe_velocity` Vector3 is received with a signal connection to velocity_computed. Avoidance processing with many registered agents has a significant performance cost and should only be enabled on agents that currently require it.

---

int **avoidance_layers** = `1` 🔗

- void **set_avoidance_layers**(value: int)
- int **get_avoidance_layers**()

A bitfield determining the avoidance layers for this NavigationAgent. Other agents with a matching bit on the avoidance_mask will avoid this agent.

---

int **avoidance_mask** = `1` 🔗

- void **set_avoidance_mask**(value: int)
- int **get_avoidance_mask**()

A bitfield determining what other avoidance agents and obstacles this NavigationAgent will avoid when a bit matches at least one of their avoidance_layers.

---

float **avoidance_priority** = `1.0` 🔗

- void **set_avoidance_priority**(value: float)
- float **get_avoidance_priority**()

The agent does not adjust the velocity for other agents that would match the avoidance_mask but have a lower avoidance_priority. This in turn makes the other agents with lower priority adjust their velocities even more to avoid collision with this agent.

---

bool **debug_enabled** = `false`

- void **set_debug_enabled**(value: bool)
- bool **get_debug_enabled**()

If `true` shows debug visuals for this agent.

Color **debug_path_custom_color** = `Color(1, 1, 1, 1)`

- void **set_debug_path_custom_color**(value: Color)
- Color **get_debug_path_custom_color**()

If debug_use_custom is `true` uses this color for this agent instead of global color.

---

float **debug_path_custom_point_size** = `4.0`

- void **set_debug_path_custom_point_size**(value: float)
- float **get_debug_path_custom_point_size**()

If debug_use_custom is `true` uses this rasterized point size for rendering path points for this agent instead of global point size.

---

bool **debug_use_custom** = `false`

- void **set_debug_use_custom**(value: bool)
- bool **get_debug_use_custom**()

If `true` uses the defined debug_path_custom_color for this agent instead of global color.

---

float **height** = `1.0`

- void **set_height**(value: float)
- float **get_height**()

The height of the avoidance agent. Agents will ignore other agents or obstacles that are above or below their current position + height in 2D avoidance. Does nothing in 3D avoidance which uses radius spheres alone.

---

bool **keep_y_velocity** = `true` 🔗

- void **set_keep_y_velocity**(value: bool)
- bool **get_keep_y_velocity**()

If `true`, and the agent uses 2D avoidance, it will remember the set y-axis velocity and reapply it after the avoidance step. While 2D avoidance has no y-axis and simulates on a flat plane this setting can help to soften the most obvious clipping on uneven 3D geometry.

---

int **max_neighbors** = `10` 🔗

- void **set_max_neighbors**(value: int)
- int **get_max_neighbors**()

The maximum number of neighbors for the agent to consider.

---

float **max_speed** = `10.0` 🔗

- void **set_max_speed**(value: float)
- float **get_max_speed**()

The maximum speed that an agent can move.

---

int **navigation_layers** = `1` 🔗

- void **set_navigation_layers**(value: int)
- int **get_navigation_layers**()

A bitfield determining which navigation layers of navigation regions this agent will use to calculate a path. Changing it during runtime will clear the current navigation path and generate a new one, according to the new navigation layers.

---

float **neighbor_distance** = `50.0` 🔗

- void **set_neighbor_distance**(value: float)
- float **get_neighbor_distance**()

The distance to search for other agents.

---

float **path_desired_distance** = `1.0` 🔗

- void **set_path_desired_distance**(value: float)
- float **get_path_desired_distance**()

The distance threshold before a path point is considered to be reached. This allows agents to not have to hit a path point on the path exactly, but only to reach its general area. If this value is set too high, the NavigationAgent will skip points on the path, which can lead to it leaving the navigation mesh. If this value is set too low, the NavigationAgent will be stuck in a repath loop because it will constantly overshoot the distance to the next point on each physics frame update.

---

float **path_height_offset** = `0.0` 🔗

- void **set_path_height_offset**(value: float)
- float **get_path_height_offset**()

The height offset is subtracted from the y-axis value of any vector path position for this NavigationAgent. The NavigationAgent height offset does not change or influence the navigation mesh or pathfinding query result. Additional navigation maps that use regions with navigation meshes that the developer baked with appropriate agent radius or height values are required to support different-sized agents.

---

float **path_max_distance** = `5.0` 🔗

- void **set_path_max_distance**(value: float)
- float **get_path_max_distance**()

The maximum distance the agent is allowed away from the ideal path to the final position. This can happen due to trying to avoid collisions. When the maximum distance is exceeded, it recalculates the ideal path.

BitField[PathMetadataFlags] **path_metadata_flags** = `7` 🔗

- void **set_path_metadata_flags**(value: BitField[PathMetadataFlags])
- BitField[PathMetadataFlags] **get_path_metadata_flags**()

Additional information to return with the navigation path.

PathPostProcessing **path_postprocessing** = `0` 🔗

- void **set_path_postprocessing**(value: PathPostProcessing)
- PathPostProcessing **get_path_postprocessing**()

The path postprocessing applied to the raw path corridor found by the pathfinding_algorithm.

PathfindingAlgorithm **pathfinding_algorithm** = `0` 🔗

- void **set_pathfinding_algorithm**(value: PathfindingAlgorithm)
- PathfindingAlgorithm **get_pathfinding_algorithm**()

The pathfinding algorithm used in the path query.

---

float **radius** = `0.5` 🔗

- void **set_radius**(value: float)
- float **get_radius**()

The radius of the avoidance agent. This is the "body" of the avoidance agent and not the avoidance maneuver starting radius (which is controlled by neighbor_distance).

Does not affect normal pathfinding. To change an actor's pathfinding radius bake NavigationMesh resources with a different NavigationMesh.agent_radius property and use different navigation maps for each actor size.

float **simplify_epsilon** = `0.0` 🔗

- void **set_simplify_epsilon**(value: float)
- float **get_simplify_epsilon**()

The path simplification amount in worlds units.

bool **simplify_path** = `false` 🔗

- void **set_simplify_path**(value: bool)
- bool **get_simplify_path**()

If `true` a simplified version of the path will be returned with less critical path points removed. The simplification amount is controlled by simplify_epsilon. The simplification uses a variant of Ramer-Douglas-Peucker algorithm for curve point decimation.

Path simplification can be helpful to mitigate various path following issues that can arise with certain agent types and script behaviors. E.g. "steering" agents or avoidance in "open fields".

float **target_desired_distance** = `1.0` 🔗

- void **set_target_desired_distance**(value: float)
- float **get_target_desired_distance**()

The distance threshold before the target is considered to be reached. On reaching the target, target_reached is emitted and navigation ends (see is_navigation_finished and navigation_finished).

You can make navigation end early by setting this property to a value greater than path_desired_distance (navigation will end before reaching the last waypoint).

You can also make navigation end closer to the target than each individual path position by setting this property to a value lower than path_desired_distance (navigation won't immediately end when reaching the last waypoint). However, if the value set is too low, the agent will be stuck in a repath loop because it will constantly overshoot the distance to the target on each physics frame update.

Vector3 **target_position** = `Vector3(0, 0, 0)`

- void **set_target_position**(value: Vector3)
- Vector3 **get_target_position**()

If set, a new navigation path from the current agent position to the target_position is requested from the NavigationServer.

---

float **time_horizon_agents** = `1.0`

- void **set_time_horizon_agents**(value: float)
- float **get_time_horizon_agents**()

The minimal amount of time for which this agent's velocities, that are computed with the collision avoidance algorithm, are safe with respect to other agents. The larger the number, the sooner the agent will respond to other agents, but less freedom in choosing its velocities. A too high value will slow down agents movement considerably. Must be positive.

---

float **time_horizon_obstacles** = `0.0`

- void **set_time_horizon_obstacles**(value: float)
- float **get_time_horizon_obstacles**()

The minimal amount of time for which this agent's velocities, that are computed with the collision avoidance algorithm, are safe with respect to static avoidance obstacles. The larger the number, the sooner the agent will respond to static avoidance obstacles, but less freedom in choosing its velocities. A too high value will slow down agents movement considerably. Must be positive.

---

bool **use_3d_avoidance** = `false` 🔗

- void **set_use_3d_avoidance**(value: bool)
- bool **get_use_3d_avoidance**()

If `true`, the agent calculates avoidance velocities in 3D omnidirectionally, e.g. for games that take place in air, underwater or space. Agents using 3D avoidance only avoid other agents using 3D avoidance, and react to radius-based avoidance obstacles. They ignore any vertex-based obstacles.

If `false`, the agent calculates avoidance velocities in 2D along the x and z-axes, ignoring the y-axis. Agents using 2D avoidance only avoid other agents using 2D avoidance, and react to radius-based avoidance obstacles or vertex-based avoidance obstacles. Other agents using 2D avoidance that are below or above their current position including height are ignored.

Vector3 **velocity** = `Vector3(0, 0, 0)` 🔗

- void **set_velocity**(value: Vector3)
- Vector3 **get_velocity**()

Sets the new wanted velocity for the agent. The avoidance simulation will try to fulfill this velocity if possible but will modify it to avoid collision with other agents and obstacles. When an agent is teleported to a new position, use set_velocity_forced as well to reset the internal simulation velocity.

Method Descriptions

float **distance_to_target**() const 🔗

Returns the distance to the target position, using the agent's global position. The user must set target_position in order for this to be accurate.

---

bool **get_avoidance_layer_value**(layer_number: int) const 🔗

Returns whether or not the specified layer of the avoidance_layers bitmask is enabled, given a `layer_number` between 1 and 32.

---

bool **get_avoidance_mask_value**(mask_number: int) const 🔗

Returns whether or not the specified mask of the avoidance_mask bitmask is enabled, given a `mask_number` between 1 and 32.

PackedVector3Array **get_current_navigation_path**() const 🔗

Returns this agent's current path from start to finish in global coordinates. The path only updates when the target position is changed or the agent requires a repath. The path array is not intended to be used in direct path movement as the agent has its own internal path logic that would get corrupted by changing the path array manually. Use the intended get_next_path_position once every physics frame to receive the next path point for the agents movement as this function also updates the internal path logic.

---

int **get_current_navigation_path_index**() const 🔗

Returns which index the agent is currently on in the navigation path's PackedVector3Array.

NavigationPathQueryResult3D **get_current_navigation_result**() const 🔗

Returns the path query result for the path the agent is currently following.

Vector3 **get_final_position**()

Returns the reachable final position of the current navigation path in global coordinates. This position can change if the agent needs to update the navigation path which makes the agent emit the path_changed signal.

bool **get_navigation_layer_value**(layer_number: int) const

Returns whether or not the specified layer of the navigation_layers bitmask is enabled, given a `layer_number` between 1 and 32.

RID **get_navigation_map**() const

Returns the RID of the navigation map for this NavigationAgent node. This function returns always the map set on the NavigationAgent node and not the map of the abstract agent on the NavigationServer. If the agent map is changed directly with the NavigationServer API the NavigationAgent node will not be aware of the map change. Use set_navigation_map to change the navigation map for the NavigationAgent and also update the agent on the NavigationServer.

Vector3 **get_next_path_position**()

Returns the next position in global coordinates that can be moved to, making sure that there are no static objects in the way. If the agent does not have a navigation path, it will return the position of the agent's parent. The use of this function once every physics frame is required to update the internal path logic of the NavigationAgent.

RID **get_rid**() const

Returns the RID of this agent on the NavigationServer3D.

bool **is_navigation_finished**() 🔗

Returns `true` if the agent's navigation has finished. If the target is reachable, navigation ends when the target is reached. If the target is unreachable, navigation ends when the last waypoint of the path is reached.

Note: While `true` prefer to stop calling update functions like get_next_path_position. This avoids jittering the standing agent due to calling repeated path updates.

---

bool **is_target_reachable**() 🔗

Returns `true` if get_final_position is within target_desired_distance of the target_position.

---

bool **is_target_reached**() const 🔗

Returns `true` if the agent reached the target, i.e. the agent moved within target_desired_distance of the target_position. It may not always be possible to reach the target but it should always be possible to reach the final position. See get_final_position.

---

void **set_avoidance_layer_value**(layer_number: int, value: bool) 🔗

Based on `value`, enables or disables the specified layer in the avoidance_layers bitmask, given a `layer_number` between 1 and 32.

---

void **set_avoidance_mask_value**(mask_number: int, value: bool) 🔗

Based on `value`, enables or disables the specified mask in the avoidance_mask bitmask, given a `mask_number` between 1 and 32.

---

void **set_navigation_layer_value**(layer_number: int, value: bool) 🔗

Based on `value`, enables or disables the specified layer in the navigation_layers bitmask, given a `layer_number` between 1 and 32.

void **set_navigation_map**(navigation_map: RID)

Sets the RID of the navigation map this NavigationAgent node should use and also updates the `agent` on the NavigationServer.

void **set_velocity_forced**(velocity: Vector3)

Replaces the internal velocity in the collision avoidance simulation with `velocity`. When an agent is teleported to a new position this function should be used in the same frame. If called frequently this function can get agents stuck.

NavigationLink2D

Experimental: This class may be changed or removed in future versions.

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A link between two positions on NavigationRegion2Ds that agents can be routed through.

Description

A link between two positions on NavigationRegion2Ds that agents can be routed through. These positions can be on the same NavigationRegion2D or on two different ones. Links are useful to express navigation methods other than traveling along the surface of the navigation polygon, such as ziplines, teleporters, or gaps that can be jumped across.

Tutorials

- Using NavigationLinks

Properties

bool	bidirectional	`true`
bool	enabled	`true`

Vector2	end_position	`Vector2(0, 0)`
float	enter_cost	`0.0`
int	navigation_layers	`1`
Vector2	start_position	`Vector2(0, 0)`
float	travel_cost	`1.0`

Methods

Vector2	get_global_end_position() const
Vector2	get_global_start_position() const
bool	get_navigation_layer_value(layer_number: int) const
RID	get_navigation_map() const
RID	get_rid() const
void	set_global_end_position(position: Vector2)

void	set_global_start_position(position: Vector2)
void	set_navigation_layer_value(layer_number: int, value: bool)
void	set_navigation_map(navigation_map: RID)

Property Descriptions

bool **bidirectional** = `true` 🔗

- void **set_bidirectional**(value: bool)
- bool **is_bidirectional**()

Whether this link can be traveled in both directions or only from start_position to end_position.

bool **enabled** = `true` 🔗

- void **set_enabled**(value: bool)
- bool **is_enabled**()

Whether this link is currently active. If `false`, NavigationServer2D.map_get_path will ignore this link.

Vector2 **end_position** = `Vector2(0, 0)` 🔗

- void **set_end_position**(value: Vector2)
- Vector2 **get_end_position**()

Ending position of the link.

This position will search out the nearest polygon in the navigation mesh to attach to.

The distance the link will search is controlled by NavigationServer2D.map_set_link_connection_radius.

---

float **enter_cost** = `0.0` 🔗

- void **set_enter_cost**(value: float)
- float **get_enter_cost**()

When pathfinding enters this link from another regions navigation mesh the enter_cost value is added to the path distance for determining the shortest path.

---

int **navigation_layers** = `1` 🔗

- void **set_navigation_layers**(value: int)
- int **get_navigation_layers**()

A bitfield determining all navigation layers the link belongs to. These navigation layers will be checked when requesting a path with NavigationServer2D.map_get_path.

Vector2 **start_position** = `Vector2(0, 0)` 🔗

- void **set_start_position**(value: Vector2)
- Vector2 **get_start_position**()

Starting position of the link.

This position will search out the nearest polygon in the navigation mesh to attach to.

The distance the link will search is controlled by NavigationServer2D.map_set_link_connection_radius.

---

float **travel_cost** = `1.0` 🔗

- void **set_travel_cost**(value: float)
- float **get_travel_cost**()

When pathfinding moves along the link the traveled distance is multiplied with travel_cost for determining the shortest path.

Method Descriptions

Vector2 **get_global_end_position**() const 🔗

Returns the end_position that is relative to the link as a global position.

Vector2 **get_global_start_position**() const 🔗

Returns the start_position that is relative to the link as a global position.

---

bool **get_navigation_layer_value**(layer_number: int) const 🔗

Returns whether or not the specified layer of the navigation_layers bitmask is enabled, given a `layer_number` between 1 and 32.

RID **get_navigation_map**() const

Returns the current navigation map RID used by this link.

RID **get_rid**() const

Returns the RID of this link on the NavigationServer2D.

void **set_global_end_position**(position: Vector2)

Sets the end_position that is relative to the link from a global `position`.

void **set_global_start_position**(position: Vector2)

Sets the start_position that is relative to the link from a global `position`.

void **set_navigation_layer_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the navigation_layers bitmask, given a `layer_number` between 1 and 32.

void **set_navigation_map**(navigation_map: RID)

Sets the RID of the navigation map this link should use. By default the link will automatically join the World2D default navigation map so this function is only required to override the default map.

NavigationLink3D

Experimental: This class may be changed or removed in future versions.

Inherits: Node3D **<** Node **<** Object

A link between two positions on NavigationRegion3Ds that agents can be routed through.

Description

A link between two positions on NavigationRegion3Ds that agents can be routed through. These positions can be on the same NavigationRegion3D or on two different ones. Links are useful to express navigation methods other than traveling along the surface of the navigation mesh, such as ziplines, teleporters, or gaps that can be jumped across.

Tutorials

- Using NavigationLinks

Properties

bool	bidirectional	`true`
bool	enabled	`true`

Vector3	end_position	`Vector3(0, 0, 0)`
float	enter_cost	`0.0`
int	navigation_layers	`1`
Vector3	start_position	`Vector3(0, 0, 0)`
float	travel_cost	`1.0`

Methods

Vector3	get_global_end_position() const
Vector3	get_global_start_position() const
bool	get_navigation_layer_value(layer_number: int) const
RID	get_navigation_map() const
RID	get_rid() const
void	set_global_end_position(position: Vector3)

void	set_global_start_position(position: Vector3)
void	set_navigation_layer_value(layer_number: int, value: bool)
void	set_navigation_map(navigation_map: RID)

Property Descriptions

bool **bidirectional** = `true` 🔗

- void **set_bidirectional**(value: bool)
- bool **is_bidirectional**()

Whether this link can be traveled in both directions or only from start_position to end_position.

---

bool **enabled** = `true` 🔗

- void **set_enabled**(value: bool)
- bool **is_enabled**()

Whether this link is currently active. If `false`, NavigationServer3D.map_get_path will ignore this link.

Vector3 **end_position** = `Vector3(0, 0, 0)` 🔗

- void **set_end_position**(value: Vector3)
- Vector3 **get_end_position**()

Ending position of the link.

This position will search out the nearest polygon in the navigation mesh to attach to.

The distance the link will search is controlled by NavigationServer3D.map_set_link_connection_radius.

---

float **enter_cost** = `0.0`

- void **set_enter_cost**(value: float)
- float **get_enter_cost**()

When pathfinding enters this link from another regions navigation mesh the enter_cost value is added to the path distance for determining the shortest path.

---

int **navigation_layers** = `1`

- void **set_navigation_layers**(value: int)
- int **get_navigation_layers**()

A bitfield determining all navigation layers the link belongs to. These navigation layers will be checked when requesting a path with NavigationServer3D.map_get_path.

Vector3 **start_position** = `Vector3(0, 0, 0)`

- void **set_start_position**(value: Vector3)
- Vector3 **get_start_position**()

Starting position of the link.

This position will search out the nearest polygon in the navigation mesh to attach to.

The distance the link will search is controlled by NavigationServer3D.map_set_link_connection_radius.

---

float **travel_cost** = `1.0`

- void **set_travel_cost**(value: float)
- float **get_travel_cost**()

When pathfinding moves along the link the traveled distance is multiplied with travel_cost for determining the shortest path.

Method Descriptions

Vector3 **get_global_end_position**() const

Returns the end_position that is relative to the link as a global position.

Vector3 **get_global_start_position**() const

Returns the start_position that is relative to the link as a global position.

---

bool **get_navigation_layer_value**(layer_number: int) const

Returns whether or not the specified layer of the navigation_layers bitmask is enabled, given a `layer_number` between 1 and 32.

RID **get_navigation_map**() const

Returns the current navigation map RID used by this link.

RID **get_rid**() const

Returns the RID of this link on the NavigationServer3D.

---

void **set_global_end_position**(position: Vector3)

Sets the end_position that is relative to the link from a global `position`.

---

void **set_global_start_position**(position: Vector3)

Sets the start_position that is relative to the link from a global `position`.

---

void **set_navigation_layer_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the navigation_layers bitmask, given a `layer_number` between 1 and 32.

---

void **set_navigation_map**(navigation_map: RID)

Sets the RID of the navigation map this link should use. By default the link will automatically join the World3D default navigation map so this function is only required to override the default map.

NavigationObstacle2D

Experimental: This class may be changed or removed in future versions.

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

2D obstacle used to affect navigation mesh baking or constrain velocities of avoidance controlled agents.

Description

An obstacle needs a navigation map and outline vertices defined to work correctly. The outlines can not cross or overlap.

Obstacles can be included in the navigation mesh baking process when affect_navigation_mesh is enabled. They do not add walkable geometry, instead their role is to discard other source geometry inside the shape. This can be used to prevent navigation mesh from appearing in unwanted places. If carve_navigation_mesh is enabled the baked shape will not be affected by offsets of the navigation mesh baking, e.g. the agent radius.

With avoidance_enabled the obstacle can constrain the avoidance velocities of avoidance using agents. If the obstacle's vertices are wound in clockwise order, avoidance agents will be pushed in by the obstacle, otherwise, avoidance agents will be pushed out. Obstacles using vertices and avoidance can warp to a new position but should not be moved every single frame as each change requires a rebuild of the avoidance map.

Tutorials

- Using NavigationObstacles

Properties

bool	affect_navigation_mesh	`false`
bool	avoidance_enabled	`true`
int	avoidance_layers	`1`
bool	carve_navigation_mesh	`false`
float	radius	`0.0`
Vector2	velocity	`Vector2(0, 0)`
PackedVector2Array	vertices	`PackedVector2Array()`

Methods

bool	get_avoidance_layer_value(layer_number: int) const
RID	get_navigation_map() const

RID	get_rid() const
void	set_avoidance_layer_value(layer_number: int, value: bool)
void	set_navigation_map(navigation_map: RID)

Property Descriptions

bool **affect_navigation_mesh** = `false`

- void **set_affect_navigation_mesh**(value: bool)
- bool **get_affect_navigation_mesh**()

If enabled and parsed in a navigation mesh baking process the obstacle will discard source geometry inside its vertices defined shape.

bool **avoidance_enabled** = `true`

- void **set_avoidance_enabled**(value: bool)
- bool **get_avoidance_enabled**()

If `true` the obstacle affects avoidance using agents.

int **avoidance_layers** = `1`

- void **set_avoidance_layers**(value: int)
- int **get_avoidance_layers**()

A bitfield determining the avoidance layers for this obstacle. Agents with a matching bit on the their avoidance mask will avoid this obstacle.

---

bool **carve_navigation_mesh** = `false`

- void **set_carve_navigation_mesh**(value: bool)
- bool **get_carve_navigation_mesh**()

If enabled the obstacle vertices will carve into the baked navigation mesh with the shape unaffected by additional offsets (e.g. agent radius).

It will still be affected by further postprocessing of the baking process, like edge and polygon simplification.

Requires affect_navigation_mesh to be enabled.

---

float **radius** = `0.0`

- void **set_radius**(value: float)
- float **get_radius**()

Sets the avoidance radius for the obstacle.

Vector2 **velocity** = `Vector2(0, 0)`

- void **set_velocity**(value: Vector2)
- Vector2 **get_velocity**()

Sets the wanted velocity for the obstacle so other agent's can better predict the obstacle if it is moved with a velocity regularly (every frame) instead of warped to a new position. Does only affect avoidance for the obstacles radius. Does nothing for the obstacles static vertices.

PackedVector2Array **vertices** = `PackedVector2Array()`

- void **set_vertices**(value: PackedVector2Array)
- PackedVector2Array **get_vertices**()

The outline vertices of the obstacle. If the vertices are winded in clockwise order agents will be pushed in by the obstacle, else they will be pushed out. Outlines can not be crossed or overlap. Should the vertices using obstacle be warped to a new position agent's can not predict this movement and may get trapped inside the obstacle.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

Method Descriptions

bool **get_avoidance_layer_value**(layer_number: int) const

Returns whether or not the specified layer of the avoidance_layers bitmask is enabled, given a `layer_number` between 1 and 32.

RID **get_navigation_map**() const

Returns the RID of the navigation map for this NavigationObstacle node. This function returns always the map set on the NavigationObstacle node and not the map of the abstract obstacle on the NavigationServer. If the obstacle map is changed directly with the NavigationServer API the NavigationObstacle node will not be aware of the map change. Use set_navigation_map to change the navigation map for the NavigationObstacle and also update the obstacle on the NavigationServer.

RID **get_rid**() const

Returns the RID of this obstacle on the NavigationServer2D.

---

void **set_avoidance_layer_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the avoidance_layers bitmask, given a `layer_number` between 1 and 32.

---

void **set_navigation_map**(navigation_map: RID)

Sets the RID of the navigation map this NavigationObstacle node should use and also updates the `obstacle` on the NavigationServer.

NavigationObstacle3D

Experimental: This class may be changed or removed in future versions.

Inherits: Node3D **<** Node **<** Object

3D obstacle used to affect navigation mesh baking or constrain velocities of avoidance controlled agents.

Description

An obstacle needs a navigation map and outline vertices defined to work correctly. The outlines can not cross or overlap and are restricted to a plane projection. This means the y-axis of the vertices is ignored, instead the obstacle's global y-axis position is used for placement. The projected shape is extruded by the obstacles height along the y-axis.

Obstacles can be included in the navigation mesh baking process when affect_navigation_mesh is enabled. They do not add walkable geometry, instead their role is to discard other source geometry inside the shape. This can be used to prevent navigation mesh from appearing in unwanted places, e.g. inside "solid" geometry or on top of it. If carve_navigation_mesh is enabled the baked shape will not be affected by offsets of the navigation mesh baking, e.g. the agent radius.

With avoidance_enabled the obstacle can constrain the avoidance velocities of avoidance using agents. If the obstacle's vertices are wound in clockwise order, avoidance agents will be pushed in by the obstacle, otherwise, avoidance agents will be pushed out. Obstacles using vertices and avoidance can warp to a new position but should not be moved every single frame as each change requires a rebuild of the avoidance map.

Tutorials

- Using NavigationObstacles

Properties

bool	affect_navigation_mesh	`false`
bool	avoidance_enabled	`true`
int	avoidance_layers	`1`
bool	carve_navigation_mesh	`false`
float	height	`1.0`
float	radius	`0.0`
bool	use_3d_avoidance	`false`
Vector3	velocity	`Vector3(0, 0, 0)`
PackedVector3Array	vertices	`PackedVector3Array()`

Methods

bool	get_avoidance_layer_value(layer_number: int) const
RID	get_navigation_map() const
RID	get_rid() const
void	set_avoidance_layer_value(layer_number: int, value: bool)
void	set_navigation_map(navigation_map: RID)

Property Descriptions

bool **affect_navigation_mesh** = `false`

- void **set_affect_navigation_mesh**(value: bool)
- bool **get_affect_navigation_mesh**()

If enabled and parsed in a navigation mesh baking process the obstacle will discard source geometry inside its vertices and height defined shape.

bool **avoidance_enabled** = `true`

- void **set_avoidance_enabled**(value: bool)

- bool **get_avoidance_enabled**()

If `true` the obstacle affects avoidance using agents.

int **avoidance_layers** = `1` 🔗

- void **set_avoidance_layers**(value: int)
- int **get_avoidance_layers**()

A bitfield determining the avoidance layers for this obstacle. Agents with a matching bit on the their avoidance mask will avoid this obstacle.

bool **carve_navigation_mesh** = `false` 🔗

- void **set_carve_navigation_mesh**(value: bool)
- bool **get_carve_navigation_mesh**()

If enabled the obstacle vertices will carve into the baked navigation mesh with the shape unaffected by additional offsets (e.g. agent radius).

It will still be affected by further postprocessing of the baking process, like edge and polygon simplification.

Requires affect_navigation_mesh to be enabled.

float **height** = `1.0` 🔗

- void **set_height**(value: float)
- float **get_height**()

Sets the obstacle height used in 2D avoidance. 2D avoidance using agent's ignore obstacles that are below or above them.

---

float **radius** = `0.0` 🔗

- void **set_radius**(value: float)
- float **get_radius**()

Sets the avoidance radius for the obstacle.

---

bool **use_3d_avoidance** = `false` 🔗

- void **set_use_3d_avoidance**(value: bool)
- bool **get_use_3d_avoidance**()

If `true` the obstacle affects 3D avoidance using agent's with obstacle radius.

If `false` the obstacle affects 2D avoidance using agent's with both obstacle vertices as well as obstacle radius.

Vector3 **velocity** = `Vector3(0, 0, 0)` 🔗

- void **set_velocity**(value: Vector3)
- Vector3 **get_velocity**()

Sets the wanted velocity for the obstacle so other agent's can better predict the obstacle if it is moved with a velocity regularly (every frame) instead of warped to a new position. Does only affect avoidance for the obstacles radius. Does nothing for the obstacles static vertices.

PackedVector3Array **vertices** = `PackedVector3Array()` 🔗

- void **set_vertices**(value: PackedVector3Array)
- PackedVector3Array **get_vertices**()

The outline vertices of the obstacle. If the vertices are winded in clockwise order agents will be pushed in by the obstacle, else they will be pushed out. Outlines can not be crossed or overlap. Should the vertices using obstacle be warped to a new position agent's can not predict this movement and may get trapped inside the obstacle.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector3Array for more details.

Method Descriptions

bool **get_avoidance_layer_value**(layer_number: int) const

Returns whether or not the specified layer of the avoidance_layers bitmask is enabled, given a `layer_number` between 1 and 32.

RID **get_navigation_map**() const

Returns the RID of the navigation map for this NavigationObstacle node. This function returns always the map set on the NavigationObstacle node and not the map of the abstract obstacle on the NavigationServer. If the obstacle map is changed directly with the NavigationServer API the NavigationObstacle node will not be aware of the map change. Use set_navigation_map to change the navigation map for the NavigationObstacle and also update the obstacle on the NavigationServer.

RID **get_rid**() const

Returns the RID of this obstacle on the NavigationServer3D.

void **set_avoidance_layer_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the avoidance_layers bitmask, given a `layer_number` between 1 and 32.

void **set_navigation_map**(navigation_map: RID)

Sets the RID of the navigation map this NavigationObstacle node should use and also updates the `obstacle` on the NavigationServer.

NavigationRegion2D

Experimental: This class may be changed or removed in future versions.

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A traversable 2D region that NavigationAgent2Ds can use for pathfinding.

Description

A traversable 2D region based on a NavigationPolygon that NavigationAgent2Ds can use for pathfinding.

Two regions can be connected to each other if they share a similar edge. You can set the minimum distance between two vertices required to connect two edges by using NavigationServer2D.map_set_edge_connection_margin.

Note: Overlapping two regions' navigation polygons is not enough for connecting two regions. They must share a similar edge.

The pathfinding cost of entering a region from another region can be controlled with the enter_cost value.

Note: This value is not added to the path cost when the start position is already inside this region.

The pathfinding cost of traveling distances inside this region can be controlled with the travel_cost multiplier.

Note: This node caches changes to its properties, so if you make changes to the underlying region RID in NavigationServer2D, they will not be reflected in this node's properties.

Tutorials

- Using NavigationRegions

Properties

bool	enabled	`true`
float	enter_cost	`0.0`
int	navigation_layers	`1`
NavigationPolygon	navigation_polygon	
float	travel_cost	`1.0`
bool	use_edge_connections	`true`

Methods

void	bake_navigation_polygon(on_thread: bool = true)
bool	get_navigation_layer_value(layer_number: int) const

RID	get_navigation_map() const
RID	get_region_rid() const
RID	get_rid() const
bool	is_baking() const
void	set_navigation_layer_value(layer_number: int, value: bool)
void	set_navigation_map(navigation_map: RID)

Signals

bake_finished()

Emitted when a navigation polygon bake operation is completed.

navigation_polygon_changed()

Emitted when the used navigation polygon is replaced or changes to the internals of the current navigation polygon are committed.

Property Descriptions

bool **enabled** = `true` 🔗

- void **set_enabled**(value: bool)
- bool **is_enabled**()

Determines if the **NavigationRegion2D** is enabled or disabled.

---

float **enter_cost** = `0.0` 🔗

- void **set_enter_cost**(value: float)
- float **get_enter_cost**()

When pathfinding enters this region's navigation mesh from another regions navigation mesh the enter_cost value is added to the path distance for determining the shortest path.

---

int **navigation_layers** = `1` 🔗

- void **set_navigation_layers**(value: int)
- int **get_navigation_layers**()

A bitfield determining all navigation layers the region belongs to. These navigation layers can be checked upon when requesting a path with NavigationServer2D.map_get_path.

NavigationPolygon **navigation_polygon** 🔗

- void **set_navigation_polygon**(value: NavigationPolygon)
- NavigationPolygon **get_navigation_polygon**()

The NavigationPolygon resource to use.

---

float **travel_cost** = `1.0` 🔗

- void **set_travel_cost**(value: float)
- float **get_travel_cost**()

When pathfinding moves inside this region's navigation mesh the traveled distances are multiplied with travel_cost for determining the shortest path.

---

bool **use_edge_connections** = `true`

- void **set_use_edge_connections**(value: bool)
- bool **get_use_edge_connections**()

If enabled the navigation region will use edge connections to connect with other navigation regions within proximity of the navigation map edge connection margin.

Method Descriptions

void **bake_navigation_polygon**(on_thread: bool = true)

Bakes the NavigationPolygon. If `on_thread` is set to `true` (default), the baking is done on a separate thread.

---

bool **get_navigation_layer_value**(layer_number: int) const

Returns whether or not the specified layer of the navigation_layers bitmask is enabled, given a `layer_number` between 1 and 32.

RID **get_navigation_map**() const

Returns the current navigation map RID used by this region.

RID **get_region_rid**() const 🔗

Deprecated: Use get_rid instead.

Returns the RID of this region on the NavigationServer2D.

RID **get_rid**() const 🔗

Returns the RID of this region on the NavigationServer2D. Combined with NavigationServer2D.map_get_closest_point_owner can be used to identify the **NavigationRegion2D** closest to a point on the merged navigation map.

---

bool **is_baking**() const 🔗

Returns `true` when the NavigationPolygon is being baked on a background thread.

---

void **set_navigation_layer_value**(layer_number: int, value: bool) 🔗

Based on `value`, enables or disables the specified layer in the navigation_layers bitmask, given a `layer_number` between 1 and 32.

---

void **set_navigation_map**(navigation_map: RID) 🔗

Sets the RID of the navigation map this region should use. By default the region will automatically join the World2D default navigation map so this function is only required to override the default map.

NavigationRegion3D

Experimental: This class may be changed or removed in future versions.

Inherits: Node3D **<** Node **<** Object

A traversable 3D region that NavigationAgent3Ds can use for pathfinding.

Description

A traversable 3D region based on a NavigationMesh that NavigationAgent3Ds can use for pathfinding.

Two regions can be connected to each other if they share a similar edge. You can set the minimum distance between two vertices required to connect two edges by using NavigationServer3D.map_set_edge_connection_margin.

Note: Overlapping two regions' navigation meshes is not enough for connecting two regions. They must share a similar edge.

The cost of entering this region from another region can be controlled with the enter_cost value.

Note: This value is not added to the path cost when the start position is already inside this region.

The cost of traveling distances inside this region can be controlled with the travel_cost multiplier.

Note: This node caches changes to its properties, so if you make changes to the underlying region RID in NavigationServer3D, they will not be reflected in this node's properties.

Tutorials

- Using NavigationRegions

Properties

bool	enabled	`true`
float	enter_cost	`0.0`
int	navigation_layers	`1`
NavigationMesh	navigation_mesh	
float	travel_cost	`1.0`
bool	use_edge_connections	`true`

Methods

void	bake_navigation_mesh(on_thread: bool = true)
bool	get_navigation_layer_value(layer_number: int) const

RID	get_navigation_map() const
RID	get_region_rid() const
RID	get_rid() const
bool	is_baking() const
void	set_navigation_layer_value(layer_number: int, value: bool)
void	set_navigation_map(navigation_map: RID)

Signals

bake_finished() 🔗

Notifies when the navigation mesh bake operation is completed.

navigation_mesh_changed() 🔗

Notifies when the NavigationMesh has changed.

Property Descriptions

bool **enabled** = `true` 🔗

- void **set_enabled**(value: bool)
- bool **is_enabled**()

Determines if the **NavigationRegion3D** is enabled or disabled.

---

float **enter_cost** = `0.0` 🔗

- void **set_enter_cost**(value: float)
- float **get_enter_cost**()

When pathfinding enters this region's navigation mesh from another regions navigation mesh the enter_cost value is added to the path distance for determining the shortest path.

---

int **navigation_layers** = `1` 🔗

- void **set_navigation_layers**(value: int)
- int **get_navigation_layers**()

A bitfield determining all navigation layers the region belongs to. These navigation layers can be checked upon when requesting a path with NavigationServer3D.map_get_path.

NavigationMesh **navigation_mesh** 🔗

- void **set_navigation_mesh**(value: NavigationMesh)
- NavigationMesh **get_navigation_mesh**()

The NavigationMesh resource to use.

---

float **travel_cost** = `1.0` 🔗

- void **set_travel_cost**(value: float)

- float **get_travel_cost**()

When pathfinding moves inside this region's navigation mesh the traveled distances are multiplied with travel_cost for determining the shortest path.

---

bool **use_edge_connections** = `true` 🔗

- void **set_use_edge_connections**(value: bool)
- bool **get_use_edge_connections**()

If enabled the navigation region will use edge connections to connect with other navigation regions within proximity of the navigation map edge connection margin.

Method Descriptions

void **bake_navigation_mesh**(on_thread: bool = true) 🔗

Bakes the NavigationMesh. If `on_thread` is set to `true` (default), the baking is done on a separate thread. Baking on separate thread is useful because navigation baking is not a cheap operation. When it is completed, it automatically sets the new NavigationMesh. Please note that baking on separate thread may be very slow if geometry is parsed from meshes as async access to each mesh involves heavy synchronization. Also, please note that baking on a separate thread is automatically disabled on operating systems that cannot use threads (such as Web with threads disabled).

---

bool **get_navigation_layer_value**(layer_number: int) const 🔗

Returns whether or not the specified layer of the navigation_layers bitmask is enabled, given a `layer_number` between 1 and 32.

RID **get_navigation_map**() const

Returns the current navigation map RID used by this region.

RID **get_region_rid**() const

Deprecated: Use get_rid instead.

Returns the RID of this region on the NavigationServer3D.

RID **get_rid**() const

Returns the RID of this region on the NavigationServer3D. Combined with NavigationServer3D.map_get_closest_point_owner can be used to identify the **NavigationRegion3D** closest to a point on the merged navigation map.

---

bool **is_baking**() const

Returns `true` when the NavigationMesh is being baked on a background thread.

---

void **set_navigation_layer_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the navigation_layers bitmask, given a `layer_number` between 1 and 32.

---

void **set_navigation_map**(navigation_map: RID)

Sets the RID of the navigation map this region should use. By default the region will automatically join the World3D default navigation map so this function is only required to override the default map.

NinePatchRect

Inherits: Control **<** CanvasItem **<** Node **<** Object

A control that displays a texture by keeping its corners intact, but tiling its edges and center.

Description

Also known as 9-slice panels, **NinePatchRect** produces clean panels of any size based on a small texture. To do so, it splits the texture in a 3×3 grid. When you scale the node, it tiles the texture's edges horizontally or vertically, tiles the center on both axes, and leaves the corners unchanged.

Properties

AxisStretchMode	axis_stretch_horizontal	`0`
AxisStretchMode	axis_stretch_vertical	`0`
bool	draw_center	`true`
MouseFilter	mouse_filter	`2` (overrides Control)
int	patch_margin_bottom	`0`

int	patch_margin_left	`0`
int	patch_margin_right	`0`
int	patch_margin_top	`0`
Rect2	region_rect	`Rect2(0, 0, 0, 0)`
Texture2D	texture	

Methods

int	get_patch_margin(margin: Side) const
void	set_patch_margin(margin: Side, value: int)

Signals

texture_changed() 🔗

Emitted when the node's texture changes.

Enumerations

enum **AxisStretchMode**: 🔗

AxisStretchMode **AXIS_STRETCH_MODE_STRETCH** = `0`

Stretches the center texture across the NinePatchRect. This may cause the texture to be distorted.

AxisStretchMode **AXIS_STRETCH_MODE_TILE** = `1`

Repeats the center texture across the NinePatchRect. This won't cause any visible distortion. The texture must be seamless for this to work without displaying artifacts between edges.

AxisStretchMode **AXIS_STRETCH_MODE_TILE_FIT** = `2`

Repeats the center texture across the NinePatchRect, but will also stretch the texture to make sure each tile is visible in full. This may cause the texture to be distorted, but less than AXIS_STRETCH_MODE_STRETCH. The texture must be seamless for this to work without displaying artifacts between edges.

Property Descriptions

AxisStretchMode **axis_stretch_horizontal** = `0` 🔗

- void **set_h_axis_stretch_mode**(value: AxisStretchMode)
- AxisStretchMode **get_h_axis_stretch_mode**()

The stretch mode to use for horizontal stretching/tiling. See AxisStretchMode for possible values.

AxisStretchMode **axis_stretch_vertical** = `0` 🔗

- void **set_v_axis_stretch_mode**(value: AxisStretchMode)

- AxisStretchMode **get_v_axis_stretch_mode**()

The stretch mode to use for vertical stretching/tiling. See AxisStretchMode for possible values.

---

bool **draw_center** = `true` 🔗

- void **set_draw_center**(value: bool)
- bool **is_draw_center_enabled**()

If `true`, draw the panel's center. Else, only draw the 9-slice's borders.

---

int **patch_margin_bottom** = `0` 🔗

- void **set_patch_margin**(margin: Side, value: int)
- int **get_patch_margin**(margin: Side) const

The height of the 9-slice's bottom row. A margin of 16 means the 9-slice's bottom corners and side will have a height of 16 pixels. You can set all 4 margin values individually to create panels with non-uniform borders.

---

int **patch_margin_left** = `0` 🔗

- void **set_patch_margin**(margin: Side, value: int)
- int **get_patch_margin**(margin: Side) const

The width of the 9-slice's left column. A margin of 16 means the 9-slice's left corners and side will have a width of 16 pixels. You can set all 4 margin values individually to create panels with non-uniform borders.

---

int **patch_margin_right** = `0` 🔗

- void **set_patch_margin**(margin: Side, value: int)
- int **get_patch_margin**(margin: Side) const

The width of the 9-slice's right column. A margin of 16 means the 9-slice's right corners and side will have a width of 16 pixels. You can set all 4 margin values individually to create panels with non-uniform borders.

---

int **patch_margin_top** = `0` 🔗

- void **set_patch_margin**(margin: Side, value: int)
- int **get_patch_margin**(margin: Side) const

The height of the 9-slice's top row. A margin of 16 means the 9-slice's top corners and side will have a height of 16 pixels. You can set all 4 margin values individually to create panels with non-uniform borders.

Rect2 **region_rect** = `Rect2(0, 0, 0, 0)` 🔗

- void **set_region_rect**(value: Rect2)
- Rect2 **get_region_rect**()

Rectangular region of the texture to sample from. If you're working with an atlas, use this property to define the area the 9-slice should use. All other properties are relative to this one. If the rect is empty, NinePatchRect will use the whole texture.

Texture2D **texture** 🔗

- void **set_texture**(value: Texture2D)
- Texture2D **get_texture**()

The node's texture resource.

Method Descriptions

int **get_patch_margin**(margin: Side) const

Returns the size of the margin on the specified Side.

---

void **set_patch_margin**(margin: Side, value: int)

Sets the size of the margin on the specified Side to `value` pixels.

Node2D

Inherits: CanvasItem **<** Node **<** Object

Inherited By: AnimatedSprite2D, AudioListener2D, AudioStreamPlayer2D, BackBufferCopy, Bone2D, Camera2D, CanvasGroup, CanvasModulate, CollisionObject2D, CollisionPolygon2D, CollisionShape2D, CPUParticles2D, GPUParticles2D, Joint2D, Light2D, LightOccluder2D, Line2D, Marker2D, MeshInstance2D, MultiMeshInstance2D, NavigationLink2D, NavigationObstacle2D, NavigationRegion2D, Parallax2D, ParallaxLayer, Path2D, PathFollow2D, Polygon2D, RayCast2D, RemoteTransform2D, ShapeCast2D, Skeleton2D, Sprite2D, TileMap, TileMapLayer, TouchScreenButton, VisibleOnScreenNotifier2D

A 2D game object, inherited by all 2D-related nodes. Has a position, rotation, scale, and Z index.

Description

A 2D game object, with a transform (position, rotation, and scale). All 2D nodes, including physics objects and sprites, inherit from Node2D. Use Node2D as a parent node to move, scale and rotate children in a 2D project. Also gives control of the node's render order.

Tutorials

- Custom drawing in 2D
- All 2D Demos [https://github.com/godotengine/godot-demo-projects/tree/master/2d]

Properties

Vector2	global_position	
float	global_rotation	
float	global_rotation_degrees	
Vector2	global_scale	
float	global_skew	
Transform2D	global_transform	
Vector2	position	`Vector2(0, 0)`
float	rotation	`0.0`
float	rotation_degrees	
Vector2	scale	`Vector2(1, 1)`
float	skew	`0.0`
Transform2D	transform	

Methods

void	apply_scale(ratio: Vector2)
float	get_angle_to(point: Vector2) const
Transform2D	get_relative_transform_to_parent(parent: Node) const
void	global_translate(offset: Vector2)
void	look_at(point: Vector2)
void	move_local_x(delta: float, scaled: bool = false)
void	move_local_y(delta: float, scaled: bool = false)
void	rotate(radians: float)
Vector2	to_global(local_point: Vector2) const
Vector2	to_local(global_point: Vector2) const
void	translate(offset: Vector2)

Property Descriptions

Vector2 **global_position** 🔗

- void **set_global_position**(value: Vector2)
- Vector2 **get_global_position**()

Global position.

Float **global_rotation** 🔗

- void **set_global_rotation**(value: float)
- float **get_global_rotation**()

Global rotation in radians.

float **global_rotation_degrees** 🔗

- void **set_global_rotation_degrees**(value: float)
- float **get_global_rotation_degrees**()

Helper property to access global_rotation in degrees instead of radians.

Vector2 **global_scale** 🔗

- void **set_global_scale**(value: Vector2)
- Vector2 **get_global_scale**()

Global scale.

float **global_skew**

- void **set_global_skew**(value: float)
- float **get_global_skew**()

Global skew in radians.

Transform2D **global_transform**

- void **set_global_transform**(value: Transform2D)
- Transform2D **get_global_transform**()

Global Transform2D.

Vector2 **position** = `Vector2(0, 0)`

- void **set_position**(value: Vector2)
- Vector2 **get_position**()

Position, relative to the node's parent.

float **rotation** = `0.0`

- void **set_rotation**(value: float)
- float **get_rotation**()

Rotation in radians, relative to the node's parent.

Note: This property is edited in the inspector in degrees. If you want to use degrees in a script, use rotation_degrees.

float **rotation_degrees**

- void **set_rotation_degrees**(value: float)
- float **get_rotation_degrees**()

Helper property to access rotation in degrees instead of radians.

Vector2 **scale** = `Vector2(1, 1)`

- void **set_scale**(value: Vector2)
- Vector2 **get_scale**()

The node's scale. Unscaled value: `(1, 1)`.

Note: Negative X scales in 2D are not decomposable from the transformation matrix. Due to the way scale is represented with transformation matrices in Godot, negative scales on the X axis will be changed to negative scales on the Y axis and a rotation of 180 degrees when decomposed.

float **skew** = `0.0`

- void **set_skew**(value: float)
- float **get_skew**()

Slants the node.

Note: Skew is X axis only.

Transform2D **transform**

- void **set_transform**(value: Transform2D)
- Transform2D **get_transform**()

Local Transform2D.

Method Descriptions

void **apply_scale**(ratio: Vector2)

Multiplies the current scale by the `ratio` vector.

---

float **get_angle_to**(point: Vector2) const

Returns the angle between the node and the `point` in radians.

Illustration of the returned angle.
[https://raw.githubusercontent.com/godotengine/godot-docs/master/img/node2d_get_angle_to.png]

Transform2D **get_relative_transform_to_parent**(parent: Node) const

Returns the Transform2D relative to this node's parent.

---

void **global_translate**(offset: Vector2)

Adds the `offset` vector to the node's global position.

---

void **look_at**(point: Vector2)

Rotates the node so that its local +X axis points towards the `point`, which is expected to use global coordinates.

`point` should not be the same as the node's position, otherwise the node always looks to the right.

---

void **move_local_x**(delta: float, scaled: bool = false) 🔗

Applies a local translation on the node's X axis based on the Node._process's `delta`. If `scaled` is `false`, normalizes the movement.

---

void **move_local_y**(delta: float, scaled: bool = false) 🔗

Applies a local translation on the node's Y axis based on the Node._process's `delta`. If `scaled` is `false`, normalizes the movement.

---

void **rotate**(radians: float) 🔗

Applies a rotation to the node, in radians, starting from its current rotation.

Vector2 **to_global**(local_point: Vector2) const 🔗

Transforms the provided local position into a position in global coordinate space. The input is expected to be local relative to the **Node2D** it is called on. e.g. Applying this method to the positions of child nodes will correctly transform their positions into the global coordinate space, but applying it to a node's own position will give an incorrect result, as it will incorporate the node's own transformation into its global position.

Vector2 **to_local**(global_point: Vector2) const 🔗

Transforms the provided global position into a position in local coordinate space. The output will be local relative to the **Node2D** it is called on. e.g. It is appropriate for determining the positions of child nodes, but it is not appropriate for determining its own position relative to its parent.

---

void **translate**(offset: Vector2) 🔗

Translates the node by the given `offset` in local coordinates.

Node3D

Inherits: Node **<** Object

Inherited By: AudioListener3D, AudioStreamPlayer3D, BoneAttachment3D, Camera3D, CollisionObject3D, CollisionPolygon3D, CollisionShape3D, GridMap, ImporterMeshInstance3D, Joint3D, LightmapProbe, Marker3D, NavigationLink3D, NavigationObstacle3D, NavigationRegion3D, OpenXRCompositionLayer, OpenXRHand, Path3D, PathFollow3D, RayCast3D, RemoteTransform3D, ShapeCast3D, Skeleton3D, SkeletonModifier3D, SpringArm3D, VehicleWheel3D, VisualInstance3D, XRFaceModifier3D, XRNode3D, XROrigin3D

Most basic 3D game object, parent of all 3D-related nodes.

Description

Most basic 3D game object, with a Transform3D and visibility settings. All other 3D game objects inherit from **Node3D**. Use **Node3D** as a parent node to move, scale, rotate and show/hide children in a 3D project.

Affine operations (rotate, scale, translate) happen in parent's local coordinate system, unless the **Node3D** object is set as top-level. Affine operations in this coordinate system correspond to direct affine operations on the **Node3D**'s transform. The word local below refers to this coordinate system. The coordinate system that is attached to the **Node3D** object itself is referred to as object-local coordinate system.

Note: Unless otherwise specified, all methods that have angle parameters must have angles specified as *radians*. To convert degrees to radians, use @GlobalScope.deg_to_rad.

Note: Be aware that "Spatial" nodes are now called "Node3D" starting with Godot 4. Any Godot 3.x references to "Spatial" nodes refer to "Node3D" in Godot 4.

Tutorials

- Introduction to 3D
- All 3D Demos [https://github.com/godotengine/godot-demo-projects/tree/master/3d]

Properties

Basis	basis	
Basis	global_basis	
Vector3	global_position	
Vector3	global_rotation	
Vector3	global_rotation_degrees	
Transform3D	global_transform	
Vector3	position	`Vector3(0, 0, 0)`

Quaternion	quaternion	
Vector3	rotation	`Vector3(0, 0, 0)`
Vector3	rotation_degrees	
RotationEditMode	rotation_edit_mode	`0`
EulerOrder	rotation_order	`2`
Vector3	scale	`Vector3(1, 1, 1)`
bool	top_level	`false`
Transform3D	transform	`Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)`
NodePath	visibility_parent	`NodePath("")`
bool	visible	`true`

Methods

void	add_gizmo(gizmo: Node3DGizmo)
void	clear_gizmos()
void	clear_subgizmo_selection()
void	force_update_transform()
Array[Node3DGizmo]	get_gizmos() const
Node3D	get_parent_node_3d() const
World3D	get_world_3d() const
void	global_rotate(axis: Vector3, angle: float)
void	global_scale(scale: Vector3)
void	global_translate(offset: Vector3)
void	hide()
bool	is_local_transform_notification_enabled() const

bool	is_scale_disabled() const
bool	is_transform_notification_enabled() const
bool	is_visible_in_tree() const
void	look_at(target: Vector3, up: Vector3 = Vector3(0, 1, 0), use_model_front: bool = false)
void	look_at_from_position(position: Vector3, target: Vector3, up: Vector3 = Vector3(0, 1, 0), use_model_front: bool = false)
void	orthonormalize()
void	rotate(axis: Vector3, angle: float)
void	rotate_object_local(axis: Vector3, angle: float)
void	rotate_x(angle: float)
void	rotate_y(angle: float)
void	rotate_z(angle: float)

void	scale_object_local(scale: Vector3)
void	set_disable_scale(disable: bool)
void	set_identity()
void	set_ignore_transform_notification(enabled: bool)
void	set_notify_local_transform(enable: bool)
void	set_notify_transform(enable: bool)
void	set_subgizmo_selection(gizmo: Node3DGizmo, id: int, transform: Transform3D)
void	show()
Vector3	to_global(local_point: Vector3) const
Vector3	to_local(global_point: Vector3) const
void	translate(offset: Vector3)
void	translate_object_local(offset: Vector3)

void	update_gizmos()

Signals

visibility_changed()

Emitted when node visibility changes.

Enumerations

enum **RotationEditMode**:

RotationEditMode **ROTATION_EDIT_MODE_EULER** = `0`

The rotation is edited using Vector3 Euler angles.

RotationEditMode **ROTATION_EDIT_MODE_QUATERNION** = `1`

The rotation is edited using a Quaternion.

RotationEditMode **ROTATION_EDIT_MODE_BASIS** = `2`

The rotation is edited using a Basis. In this mode, scale can't be edited separately.

Constants

NOTIFICATION_TRANSFORM_CHANGED = `2000`

Node3D nodes receive this notification when their global transform changes. This means that either the current or a parent node changed its transform.

In order for NOTIFICATION_TRANSFORM_CHANGED to work, users first need to ask for it, with set_notify_transform. The notification is also sent if the node is in the editor context and it has at least one valid gizmo.

NOTIFICATION_ENTER_WORLD = `41` 🔗

Node3D nodes receive this notification when they are registered to new World3D resource.

NOTIFICATION_EXIT_WORLD = `42` 🔗

Node3D nodes receive this notification when they are unregistered from current World3D resource.

NOTIFICATION_VISIBILITY_CHANGED = `43` 🔗

Node3D nodes receive this notification when their visibility changes.

NOTIFICATION_LOCAL_TRANSFORM_CHANGED = `44` 🔗

Node3D nodes receive this notification when their local transform changes. This is not received when the transform of a parent node is changed.

In order for NOTIFICATION_LOCAL_TRANSFORM_CHANGED to work, users first need to ask for it, with set_notify_local_transform.

Property Descriptions

Basis **basis** 🔗

- void **set_basis**(value: Basis)
- Basis **get_basis**()

Basis of the transform property. Represents the rotation, scale, and shear of this node.

Basis **global_basis** 🔗

- void **set_global_basis**(value: Basis)
- Basis **get_global_basis**()

Global basis of this node. This is equivalent to `global_transform.basis`.

Vector3 **global_position** 🔗

- void **set_global_position**(value: Vector3)
- Vector3 **get_global_position**()

Global position of this node. This is equivalent to `global_transform.origin`.

Vector3 **global_rotation** 🔗

- void **set_global_rotation**(value: Vector3)
- Vector3 **get_global_rotation**()

Rotation part of the global transformation in radians, specified in terms of YXZ-Euler angles in the format (X angle, Y angle, Z angle).

Note: In the mathematical sense, rotation is a matrix and not a vector. The three Euler angles, which are the three independent parameters of the Euler-angle parametrization of the rotation matrix, are stored in a Vector3 data structure not because the rotation is a vector, but only because Vector3 exists as a convenient data-structure to store 3 floating-point numbers. Therefore, applying affine operations on the rotation "vector" is not meaningful.

Vector3 **global_rotation_degrees** 🔗

- void **set_global_rotation_degrees**(value: Vector3)
- Vector3 **get_global_rotation_degrees**()

Helper property to access global_rotation in degrees instead of radians.

Transform3D **global_transform** 🔗

- void **set_global_transform**(value: Transform3D)
- Transform3D **get_global_transform**()

World3D space (global) Transform3D of this node.

Vector3 **position** = `Vector3(0, 0, 0)` 🔗

- void **set_position**(value: Vector3)
- Vector3 **get_position**()

Local position or translation of this node relative to the parent. This is equivalent to `transform.origin`.

Quaternion **quaternion** 🔗

- void **set_quaternion**(value: Quaternion)
- Quaternion **get_quaternion**()

Access to the node rotation as a Quaternion. This property is ideal for tweening complex rotations.

Vector3 **rotation** = `Vector3(0, 0, 0)` 🔗

- void **set_rotation**(value: Vector3)
- Vector3 **get_rotation**()

Rotation part of the local transformation in radians, specified in terms of Euler angles. The angles construct a rotation in the order specified by the rotation_order property.

Note: In the mathematical sense, rotation is a matrix and not a vector. The three Euler angles, which are the three independent parameters of the Euler-angle parametrization of the rotation matrix, are stored in a Vector3 data structure not because the rotation is a vector, but only because Vector3 exists as a convenient data-structure to store 3 floating-point numbers. Therefore, applying affine operations on the rotation "vector" is not meaningful.

Note: This property is edited in the inspector in degrees. If you want to use degrees in a script, use rotation_degrees.

Vector3 **rotation_degrees** 🔗

- void **set_rotation_degrees**(value: Vector3)
- Vector3 **get_rotation_degrees**()

Helper property to access rotation in degrees instead of radians.

RotationEditMode **rotation_edit_mode** = `0` 🔗

- void **set_rotation_edit_mode**(value: RotationEditMode)
- RotationEditMode **get_rotation_edit_mode**()

Specify how rotation (and scale) will be presented in the editor.

EulerOrder **rotation_order** = `2` 🔗

- void **set_rotation_order**(value: EulerOrder)
- EulerOrder **get_rotation_order**()

Specify the axis rotation order of the rotation property. The final orientation is constructed by rotating the Euler angles in the order specified by this property.

Vector3 **scale** = `Vector3(1, 1, 1)` 🔗

- void **set_scale**(value: Vector3)
- Vector3 **get_scale**()

Scale part of the local transformation.

Note: Mixed negative scales in 3D are not decomposable from the transformation matrix. Due to the way scale is represented with transformation matrices in Godot, the scale values will either be all positive or all negative.

Note: Not all nodes are visually scaled by the scale property. For example, Light3Ds are not visually affected by scale.

---

bool **top_level** = `false` 🔗

- void **set_as_top_level**(value: bool)
- bool **is_set_as_top_level**()

If `true`, the node will not inherit its transformations from its parent. Node transformations are only in global space.

Transform3D **transform** = `Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)` 🔗

- void **set_transform**(value: Transform3D)
- Transform3D **get_transform**()

Local space Transform3D of this node, with respect to the parent node.

NodePath **visibility_parent** = `NodePath("")` 🔗

- void **set_visibility_parent**(value: NodePath)
- NodePath **get_visibility_parent**()

Defines the visibility range parent for this node and its subtree. The visibility parent must be a GeometryInstance3D. Any visual instance will only be visible if the visibility parent (and all of its visibility ancestors) is hidden by being closer to the camera than its own GeometryInstance3D.visibility_range_begin. Nodes hidden via the visible property are essentially removed from the visibility dependency tree, so dependent instances will not take the hidden node or its ancestors into account.

---

bool **visible** = `true` 🔗

- void **set_visible**(value: bool)
- bool **is_visible**()

If `true`, this node is drawn. The node is only visible if all of its ancestors are visible as well (in other words, is_visible_in_tree must return `true`).

Method Descriptions

void **add_gizmo**(gizmo: Node3DGizmo) 🔗

Attach an editor gizmo to this **Node3D**.

Note: The gizmo object would typically be an instance of EditorNode3DGizmo, but the argument type is kept generic to avoid creating a dependency on editor classes in **Node3D**.

void **clear_gizmos**() 🔗

Clear all gizmos attached to this **Node3D**.

void **clear_subgizmo_selection**() 🔗

Clears subgizmo selection for this node in the editor. Useful when subgizmo IDs become invalid after a property change.

void **force_update_transform**() 🔗

Forces the transform to update. Transform changes in physics are not instant for performance reasons. Transforms are accumulated and then set. Use this if you need an up-to-date transform when doing physics operations.

Array[Node3DGizmo] **get_gizmos**() const 🔗

Returns all the gizmos attached to this **Node3D**.

Node3D **get_parent_node_3d**() const 🔗

Returns the parent **Node3D**, or `null` if no parent exists, the parent is not of type **Node3D**, or top_level is `true`.

Note: Calling this method is not equivalent to `get_parent() as Node3D`, which does not take top_level into account.

World3D **get_world_3d**() const 🔗

Returns the current World3D resource this **Node3D** node is registered to.

void **global_rotate**(axis: Vector3, angle: float) 🔗

Rotates the global (world) transformation around axis, a unit Vector3, by specified angle in radians. The rotation axis is in global coordinate system.

void **global_scale**(scale: Vector3) 🔗

Scales the global (world) transformation by the given Vector3 scale factors.

void **global_translate**(offset: Vector3) 🔗

Moves the global (world) transformation by Vector3 offset. The offset is in global coordinate system.

void **hide**() 🔗

Disables rendering of this node. Changes visible to `false`.

bool **is_local_transform_notification_enabled**() const 🔗

Returns whether node notifies about its local transformation changes. **Node3D** will not propagate this by default.

bool **is_scale_disabled**() const

Returns whether this node uses a scale of `(1, 1, 1)` or its local transformation scale.

bool **is_transform_notification_enabled**() const

Returns whether the node notifies about its global and local transformation changes. **Node3D** will not propagate this by default.

bool **is_visible_in_tree**() const

Returns `true` if the node is present in the SceneTree, its visible property is `true` and all its ancestors are also visible. If any ancestor is hidden, this node will not be visible in the scene tree.

void **look_at**(target: Vector3, up: Vector3 = Vector3(0, 1, 0), use_model_front: bool = false)

Rotates the node so that the local forward axis (-Z, Vector3.FORWARD) points toward the `target` position.

The local up axis (+Y) points as close to the `up` vector as possible while staying perpendicular to the local forward axis. The resulting transform is orthogonal, and the scale is preserved. Non-uniform scaling may not work correctly.

The `target` position cannot be the same as the node's position, the `up` vector cannot be zero, and the direction from the node's position to the `target` vector cannot be parallel to the `up` vector.

Operations take place in global space, which means that the node must be in the scene tree.

If `use_model_front` is `true`, the +Z axis (asset front) is treated as forward (implies +X is left) and points toward the `target` position. By default, the -Z axis (camera forward) is treated as forward (implies +X is right).

---

void **look_at_from_position**(position: Vector3, target: Vector3, up: Vector3 = Vector3(0, 1, 0), use_model_front: bool = false) 🔗

Moves the node to the specified `position`, and then rotates the node to point toward the `target` as per look_at. Operations take place in global space.

---

void **orthonormalize**() 🔗

Resets this node's transformations (like scale, skew and taper) preserving its rotation and translation by performing Gram-Schmidt orthonormalization on this node's Transform3D.

---

void **rotate**(axis: Vector3, angle: float) 🔗

Rotates the local transformation around axis, a unit Vector3, by specified angle in radians.

---

void **rotate_object_local**(axis: Vector3, angle: float) 🔗

Rotates the local transformation around axis, a unit Vector3, by specified angle in radians. The rotation axis is in object-local coordinate system.

---

void **rotate_x**(angle: float)

Rotates the local transformation around the X axis by angle in radians.

---

void **rotate_y**(angle: float)

Rotates the local transformation around the Y axis by angle in radians.

---

void **rotate_z**(angle: float)

Rotates the local transformation around the Z axis by angle in radians.

---

void **scale_object_local**(scale: Vector3)

Scales the local transformation by given 3D scale factors in object-local coordinate system.

---

void **set_disable_scale**(disable: bool)

Sets whether the node uses a scale of `(1, 1, 1)` or its local transformation scale. Changes to the local transformation scale are preserved.

---

void **set_identity**()

Reset all transformations for this node (sets its Transform3D to the identity matrix).

---

void **set_ignore_transform_notification**(enabled: bool) 🔗

Sets whether the node ignores notification that its transformation (global or local) changed.

---

void **set_notify_local_transform**(enable: bool) 🔗

Sets whether the node notifies about its local transformation changes. **Node3D** will not propagate this by default.

---

void **set_notify_transform**(enable: bool) 🔗

Sets whether the node notifies about its global and local transformation changes. **Node3D** will not propagate this by default, unless it is in the editor context and it has a valid gizmo.

---

void **set_subgizmo_selection**(gizmo: Node3DGizmo, id: int, transform: Transform3D) 🔗

Set subgizmo selection for this node in the editor.

Note: The gizmo object would typically be an instance of EditorNode3DGizmo, but the argument type is kept generic to avoid creating a dependency on editor classes in **Node3D**.

---

void **show**() 🔗

Enables rendering of this node. Changes visible to `true`.

Vector3 **to_global**(local_point: Vector3) const

Transforms `local_point` from this node's local space to world space.

Vector3 **to_local**(global_point: Vector3) const

Transforms `global_point` from world space to this node's local space.

void **translate**(offset: Vector3)

Changes the node's position by the given offset Vector3.

Note that the translation `offset` is affected by the node's scale, so if scaled by e.g. `(10, 1, 1)`, a translation by an offset of `(2, 0, 0)` would actually add 20 (`2 * 10`) to the X coordinate.

void **translate_object_local**(offset: Vector3)

Changes the node's position by the given offset Vector3 in local space.

void **update_gizmos**()

Updates all the **Node3D** gizmos attached to this node.

OccluderInstance3D

Inherits: VisualInstance3D **<** Node3D **<** Node **<** Object

Provides occlusion culling for 3D nodes, which improves performance in closed areas.

Description

Occlusion culling can improve rendering performance in closed/semi-open areas by hiding geometry that is occluded by other objects.

The occlusion culling system is mostly static. **OccluderInstance3D**s can be moved or hidden at run-time, but doing so will trigger a background recomputation that can take several frames. It is recommended to only move **OccluderInstance3D**s sporadically (e.g. for procedural generation purposes), rather than doing so every frame.

The occlusion culling system works by rendering the occluders on the CPU in parallel using Embree [https://www.embree.org/], drawing the result to a low-resolution buffer then using this to cull 3D nodes individually. In the 3D editor, you can preview the occlusion culling buffer by choosing **Perspective > Debug Advanced... > Occlusion Culling Buffer** in the top-left corner of the 3D viewport. The occlusion culling buffer quality can be adjusted in the Project Settings.

Baking: Select an **OccluderInstance3D** node, then use the **Bake Occluders** button at the top of the 3D editor. Only opaque materials will be taken into account; transparent materials (alpha-blended or alpha-tested) will be ignored by the occluder generation.

Note: Occlusion culling is only effective if ProjectSettings.rendering/occlusion_culling/use_occlusion_culling is `true`. Enabling occlusion culling has a cost on the CPU. Only enable occlusion culling if you actually plan to use it. Large open scenes with few or no objects blocking the view will generally not benefit much from occlusion culling. Large open scenes generally benefit more from mesh LOD and visibility ranges (GeometryInstance3D.visibility_range_begin and GeometryInstance3D.visibility_range_end) compared to occlusion culling.

Note: Due to memory constraints, occlusion culling is not supported by default in Web export templates. It can be enabled by compiling custom Web export templates with `module_raycast_enabled=yes`.

Tutorials

- Occlusion culling

Properties

int	bake_mask	`4294967295`
float	bake_simplification_distance	`0.1`
Occluder3D	occluder	

Methods

bool	get_bake_mask_value(layer_number: int) const
void	set_bake_mask_value(layer_number: int, value: bool)

Property Descriptions

int **bake_mask** = `4294967295`

- void **set_bake_mask**(value: int)
- int **get_bake_mask**()

The visual layers to account for when baking for occluders. Only MeshInstance3Ds whose VisualInstance3D.layers match with this bake_mask will be included in the generated occluder mesh. By default, all objects with *opaque* materials are taken into account for the occluder baking.

To improve performance and avoid artifacts, it is recommended to exclude dynamic objects, small objects and fixtures from the baking process by moving them to a separate visual layer and excluding this layer in bake_mask.

---

float **bake_simplification_distance** = `0.1`

- void **set_bake_simplification_distance**(value: float)
- float **get_bake_simplification_distance**()

The simplification distance to use for simplifying the generated occluder polygon (in 3D units). Higher values result in a less detailed occluder mesh, which improves performance but reduces culling accuracy.

The occluder geometry is rendered on the CPU, so it is important to keep its geometry as simple as possible. Since the buffer is rendered at a low resolution, less detailed occluder meshes generally still work well. The default value is fairly aggressive, so you may have to decrease it if you run into false negatives (objects being occluded even though they are visible by the camera). A value of `0.01` will act conservatively, and will keep geometry *perceptually* unaffected in the occlusion culling buffer. Depending on the scene, a value of `0.01` may still simplify the mesh noticeably compared to disabling simplification entirely.

Setting this to `0.0` disables simplification entirely, but vertices in the exact same position will still be merged. The mesh will also be re-indexed to reduce both the number of vertices and indices.

Note: This uses the meshoptimizer [https://meshoptimizer.org/] library under the hood, similar to LOD generation.

Occluder3D **occluder** 🔗

- void **set_occluder**(value: Occluder3D)
- Occluder3D **get_occluder**()

The occluder resource for this **OccluderInstance3D**. You can generate an occluder resource by selecting an **OccluderInstance3D** node then using the **Bake Occluders** button at the top of the editor.

You can also draw your own 2D occluder polygon by adding a new PolygonOccluder3D resource to the occluder property in the Inspector.

Alternatively, you can select a primitive occluder to use: QuadOccluder3D, BoxOccluder3D or SphereOccluder3D.

Method Descriptions

bool **get_bake_mask_value**(layer_number: int) const 🔗

Returns whether or not the specified layer of the bake_mask is enabled, given a `layer_number` between 1 and 32.

---

void **set_bake_mask_value**(layer_number: int, value: bool) 🔗

Based on `value`, enables or disables the specified layer in the bake_mask, given a `layer_number` between 1 and 32.

OmniLight3D

Inherits: Light3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

Omnidirectional light, such as a light bulb or a candle.

Description

An Omnidirectional light is a type of Light3D that emits light in all directions. The light is attenuated by distance and this attenuation can be configured by changing its energy, radius, and attenuation parameters.

Note: When using the Mobile rendering method, only 8 omni lights can be displayed on each mesh resource. Attempting to display more than 8 omni lights on a single mesh resource will result in omni lights flickering in and out as the camera moves. When using the Compatibility rendering method, only 8 omni lights can be displayed on each mesh resource by default, but this can be increased by adjusting ProjectSettings.rendering/limits/opengl/max_lights_per_object.

Note: When using the Mobile or Compatibility rendering methods, omni lights will only correctly affect meshes whose visibility AABB intersects with the light's AABB. If using a shader to deform the mesh in a way that makes it go outside its AABB, GeometryInstance3D.extra_cull_margin must be increased on the mesh. Otherwise, the light may not be visible on the mesh.

Tutorials

- 3D lights and shadows

- Faking global illumination

Properties

float	omni_attenuation	`1.0`
float	omni_range	`5.0`
ShadowMode	omni_shadow_mode	`1`
float	shadow_normal_bias	`1.0` (overrides Light3D)

Enumerations

enum **ShadowMode**: 🔗

ShadowMode **SHADOW_DUAL_PARABOLOID** = `0`

Shadows are rendered to a dual-paraboloid texture. Faster than SHADOW_CUBE, but lower-quality.

ShadowMode **SHADOW_CUBE** = `1`

Shadows are rendered to a cubemap. Slower than SHADOW_DUAL_PARABOLOID, but higher-quality.

Property Descriptions

float **omni_attenuation** = `1.0`

- void **set_param**(value: float)
- float **get_param**()

Controls the distance attenuation function for omnilights.

A value of `0.0` will maintain a constant brightness through most of the range, but smoothly attenuate the light at the edge of the range. Use a value of `2.0` for physically accurate lights as it results in the proper inverse square attenutation.

Note: Setting attenuation to `2.0` or higher may result in distant objects receiving minimal light, even within range. For example, with a range of `4096`, an object at `100` units is attenuated by a factor of `0.0001`. With a default brightness of `1`, the light would not be visible at that distance.

Note: Using negative or values higher than `10.0` may lead to unexpected results.

---

float **omni_range** = `5.0`

- void **set_param**(value: float)
- float **get_param**()

The light's radius. Note that the effectively lit area may appear to be smaller depending on the omni_attenuation in use. No matter the omni_attenuation in use, the light will never reach anything outside this radius.

Note: omni_range is not affected by Node3D.scale (the light's scale or its parent's scale).

ShadowMode **omni_shadow_mode** = `1`

- void **set_shadow_mode**(value: ShadowMode)
- ShadowMode **get_shadow_mode**()

See ShadowMode.

OpenXRCompositionLayer

Experimental: This class may be changed or removed in future versions.

Inherits: Node3D **<** Node **<** Object

Inherited By: OpenXRCompositionLayerCylinder, OpenXRCompositionLayerEquirect, OpenXRCompositionLayerQuad

The parent class of all OpenXR composition layer nodes.

Description

Composition layers allow 2D viewports to be displayed inside of the headset by the XR compositor through special projections that retain their quality. This allows for rendering clear text while keeping the layer at a native resolution.

Note: If the OpenXR runtime doesn't support the given composition layer type, a fallback mesh can be generated with a ViewportTexture, in order to emulate the composition layer.

Properties

bool	alpha_blend	`false`
bool	enable_hole_punch	`false`

SubViewport	layer_viewport	
int	sort_order	`1`

Methods

Vector2	intersects_ray(origin: Vector3, direction: Vector3) const
bool	is_natively_supported() const

Property Descriptions

bool **alpha_blend** = `false` 🔗

- void **set_alpha_blend**(value: bool)
- bool **get_alpha_blend**()

Enables the blending the layer using its alpha channel.

Can be combined with Viewport.transparent_bg to give the layer a transparent background.

bool **enable_hole_punch** = `false` 🔗

- void **set_enable_hole_punch**(value: bool)
- bool **get_enable_hole_punch**()

Enables a technique called "hole punching", which allows putting the composition layer behind the main projection layer (i.e. setting sort_order to a negative value) while "punching a hole" through everything rendered by Godot so that the layer is still visible.

This can be used to create the illusion that the composition layer exists in the same 3D space as everything rendered by Godot, allowing objects to appear to pass both behind or in front of the composition layer.

SubViewport **layer_viewport** 🔗

- void **set_layer_viewport**(value: SubViewport)
- SubViewport **get_layer_viewport**()

The SubViewport to render on the composition layer.

---

int **sort_order** = `1` 🔗

- void **set_sort_order**(value: int)
- int **get_sort_order**()

The sort order for this composition layer. Higher numbers will be shown in front of lower numbers.

Note: This will have no effect if a fallback mesh is being used.

Method Descriptions

Vector2 **intersects_ray**(origin: Vector3, direction: Vector3) const 🔗

Returns UV coordinates where the given ray intersects with the composition layer. `origin` and `direction` must be in global space.

Returns `Vector2(-1.0, -1.0)` if the ray doesn't intersect.

---

bool **is_natively_supported**() const 🔗

Returns true if the OpenXR runtime natively supports this composition layer type.

Note: This will only return an accurate result after the OpenXR session has started.

OpenXRCompositionLayerCylinder

Experimental: This class may be changed or removed in future versions.

Inherits: OpenXRCompositionLayer **<** Node3D **<** Node **<** Object

An OpenXR composition layer that is rendered as an internal slice of a cylinder.

Description

An OpenXR composition layer that allows rendering a SubViewport on an internal slice of a cylinder.

Properties

float	aspect_ratio	`1.0`
float	central_angle	`1.5708`
int	fallback_segments	`10`
float	radius	`1.0`

Property Descriptions

float **aspect_ratio** = `1.0`

- void **set_aspect_ratio**(value: float)
- float **get_aspect_ratio**()

The aspect ratio of the slice. Used to set the height relative to the width.

---

float **central_angle** = `1.5708`

- void **set_central_angle**(value: float)
- float **get_central_angle**()

The central angle of the cylinder. Used to set the width.

---

int **fallback_segments** = `10`

- void **set_fallback_segments**(value: int)
- int **get_fallback_segments**()

The number of segments to use in the fallback mesh.

---

float **radius** = `1.0`

- void **set_radius**(value: float)
- float **get_radius**()

The radius of the cylinder.

OpenXRCompositionLayerEquirect

Experimental: This class may be changed or removed in future versions.

Inherits: OpenXRCompositionLayer **<** Node3D **<** Node **<** Object

An OpenXR composition layer that is rendered as an internal slice of a sphere.

Description

An OpenXR composition layer that allows rendering a SubViewport on an internal slice of a sphere.

Properties

float	central_horizontal_angle	`1.5708`
int	fallback_segments	`10`
float	lower_vertical_angle	`0.785398`
float	radius	`1.0`

float	upper_vertical_angle	`0.785398`

Property Descriptions

float **central_horizontal_angle** = `1.5708`

- void **set_central_horizontal_angle**(value: float)
- float **get_central_horizontal_angle**()

The central horizontal angle of the sphere. Used to set the width.

int **fallback_segments** = `10`

- void **set_fallback_segments**(value: int)
- int **get_fallback_segments**()

The number of segments to use in the fallback mesh.

float **lower_vertical_angle** = `0.785398`

- void **set_lower_vertical_angle**(value: float)
- float **get_lower_vertical_angle**()

The lower vertical angle of the sphere. Used (together with upper_vertical_angle) to set the height.

float **radius** = `1.0`

- void **set_radius**(value: float)
- float **get_radius**()

The radius of the sphere.

---

float **upper_vertical_angle** = `0.785398` 🔗

- void **set_upper_vertical_angle**(value: float)
- float **get_upper_vertical_angle**()

The upper vertical angle of the sphere. Used (together with lower_vertical_angle) to set the height.

OpenXRCompositionLayerQuad

Experimental: This class may be changed or removed in future versions.

Inherits: OpenXRCompositionLayer **<** Node3D **<** Node **<** Object

An OpenXR composition layer that is rendered as a quad.

Description

An OpenXR composition layer that allows rendering a SubViewport on a quad.

Properties

Vector2	quad_size	`Vector2(1, 1)`

Property Descriptions

Vector2 **quad_size** = `Vector2(1, 1)` 🔗

- void **set_quad_size**(value: Vector2)
- Vector2 **get_quad_size**()

The dimensions of the quad.

OpenXRHand

Deprecated: Use XRHandModifier3D instead.

Inherits: Node3D **<** Node **<** Object

Node supporting hand and finger tracking in OpenXR.

Description

This node enables OpenXR's hand tracking functionality. The node should be a child node of an XROrigin3D node, tracking will update its position to the player's tracked hand Palm joint location (the center of the middle finger's metacarpal bone). This node also updates the skeleton of a properly skinned hand or avatar model.

If the skeleton is a hand (one of the hand bones is the root node of the skeleton), then the skeleton will be placed relative to the hand palm location and the hand mesh and skeleton should be children of the OpenXRHand node.

If the hand bones are part of a full skeleton, then the root of the hand will keep its location with the assumption that IK is used to position the hand and arm.

By default the skeleton hand bones are repositioned to match the size of the tracked hand. To preserve the modeled bone sizes change bone_update to apply rotation only.

Properties

BoneUpdate	bone_update	`0`
Hands	hand	`0`
NodePath	hand_skeleton	`NodePath("")`
MotionRange	motion_range	`0`
SkeletonRig	skeleton_rig	`0`

Enumerations

enum **Hands**: 🔗

Hands **HAND_LEFT** = `0`

Tracking the player's left hand.

Hands **HAND_RIGHT** = `1`

Tracking the player's right hand.

Hands **HAND_MAX** = `2`

Maximum supported hands.

enum **MotionRange**: 🔗

MotionRange **MOTION_RANGE_UNOBSTRUCTED** = `0`

When player grips, hand skeleton will form a full fist.

MotionRange **MOTION_RANGE_CONFORM_TO_CONTROLLER** = `1`

When player grips, hand skeleton conforms to the controller the player is holding.

MotionRange **MOTION_RANGE_MAX** = `2`

Maximum supported motion ranges.

enum **SkeletonRig**: 🔗

SkeletonRig **SKELETON_RIG_OPENXR** = `0`

An OpenXR compliant skeleton.

SkeletonRig **SKELETON_RIG_HUMANOID** = `1`

A SkeletonProfileHumanoid compliant skeleton.

SkeletonRig **SKELETON_RIG_MAX** = `2`

Maximum supported hands.

enum **BoneUpdate**: 🔗

BoneUpdate **BONE_UPDATE_FULL** = `0`

The skeletons bones are fully updated (both position and rotation) to match the tracked bones.

BoneUpdate **BONE_UPDATE_ROTATION_ONLY** = `1`

The skeletons bones are only rotated to align with the tracked bones, preserving bone length.

BoneUpdate **BONE_UPDATE_MAX** = `2`

Maximum supported bone update mode.

Property Descriptions

BoneUpdate **bone_update** = `0` 🔗

- void **set_bone_update**(value: BoneUpdate)
- BoneUpdate **get_bone_update**()

Specify the type of updates to perform on the bone.

Hands **hand** = `0` 🔗

- void **set_hand**(value: Hands)
- Hands **get_hand**()

Specifies whether this node tracks the left or right hand of the player.

NodePath **hand_skeleton** = `NodePath("")` 🔗

- void **set_hand_skeleton**(value: NodePath)
- NodePath **get_hand_skeleton**()

Set a Skeleton3D node for which the pose positions will be updated.

MotionRange **motion_range** = `0` 🔗

- void **set_motion_range**(value: MotionRange)
- MotionRange **get_motion_range**()

Set the motion range (if supported) limiting the hand motion.

SkeletonRig **skeleton_rig** = `0` 🔗

- void **set_skeleton_rig**(value: SkeletonRig)
- SkeletonRig **get_skeleton_rig**()

Set the type of skeleton rig the hand_skeleton is compliant with.

OptionButton

Inherits: Button **<** BaseButton **<** Control **<** CanvasItem **<** Node **<** Object

A button that brings up a dropdown with selectable options when pressed.

Description

OptionButton is a type of button that brings up a dropdown with selectable items when pressed. The item selected becomes the "current" item and is displayed as the button text.

See also BaseButton which contains common properties and methods associated with this node.

Note: The ID values used for items are limited to 32 bits, not full 64 bits of int. This has a range of `-2^32` to `2^32 - 1`, i.e. `-2147483648` to `2147483647`.

Note: The Button.text and Button.icon properties are set automatically based on the selected item. They shouldn't be changed manually.

Properties

ActionMode	action_mode	`0` (overrides BaseButton)
HorizontalAlignment	alignment	`0` (overrides Button)

bool	allow_reselect	`false`
bool	fit_to_longest_item	`true`
int	item_count	`0`
int	selected	`-1`
bool	toggle_mode	`true` (overrides BaseButton)

Methods

void	add_icon_item(texture: Texture2D, label: String, id: int = -1)
void	add_item(label: String, id: int = -1)
void	add_separator(text: String = "")
void	clear()
Texture2D	get_item_icon(idx: int) const

int	get_item_id(idx: int) const
int	get_item_index(id: int) const
Variant	get_item_metadata(idx: int) const
String	get_item_text(idx: int) const
String	get_item_tooltip(idx: int) const
PopupMenu	get_popup() const
int	get_selectable_item(from_last: bool = false) const
int	get_selected_id() const
Variant	get_selected_metadata() const
bool	has_selectable_items() const
bool	is_item_disabled(idx: int) const
bool	is_item_separator(idx: int) const
void	remove_item(idx: int)

void	select(idx: int)
void	set_disable_shortcuts(disabled: bool)
void	set_item_disabled(idx: int, disabled: bool)
void	set_item_icon(idx: int, texture: Texture2D)
void	set_item_id(idx: int, id: int)
void	set_item_metadata(idx: int, metadata: Variant)
void	set_item_text(idx: int, text: String)
void	set_item_tooltip(idx: int, tooltip: String)
void	show_popup()

Theme Properties

int	arrow_margin	4
int	modulate_arrow	0

Texture2D arrow

Signals

item_focused(index: int)

Emitted when the user navigates to an item using the ProjectSettings.input/ui_up or ProjectSettings.input/ui_down input actions. The index of the item selected is passed as argument.

---

item_selected(index: int)

Emitted when the current item has been changed by the user. The index of the item selected is passed as argument.

allow_reselect must be enabled to reselect an item.

Property Descriptions

bool **allow_reselect** = `false`

- void **set_allow_reselect**(value: bool)
- bool **get_allow_reselect**()

If `true`, the currently selected item can be selected again.

---

bool **fit_to_longest_item** = `true`

- void **set_fit_to_longest_item**(value: bool)

- bool **is_fit_to_longest_item**()

If `true`, minimum size will be determined by the longest item's text, instead of the currently selected one's.

Note: For performance reasons, the minimum size doesn't update immediately when adding, removing or modifying items.

---

int **item_count** = `0` 🔗

- void **set_item_count**(value: int)
- int **get_item_count**()

The number of items to select from.

---

int **selected** = `-1` 🔗

- int **get_selected**()

The index of the currently selected item, or `-1` if no item is selected.

Method Descriptions

void **add_icon_item**(texture: Texture2D, label: String, id: int = -1) 🔗

Adds an item, with a `texture` icon, text `label` and (optionally) `id`. If no `id` is passed, the item index will be used as the item's ID. New items are appended at the end.

---

void **add_item**(label: String, id: int = -1) 🔗

Adds an item, with text `label` and (optionally) `id`. If no `id` is passed, the item index will be used as the item's ID. New items are appended at the end.

---

void **add_separator**(text: String = "")

Adds a separator to the list of items. Separators help to group items, and can optionally be given a `text` header. A separator also gets an index assigned, and is appended at the end of the item list.

---

void **clear**()

Clears all the items in the **OptionButton**.

Texture2D **get_item_icon**(idx: int) const

Returns the icon of the item at index `idx`.

---

int **get_item_id**(idx: int) const

Returns the ID of the item at index `idx`.

---

int **get_item_index**(id: int) const

Returns the index of the item with the given `id`.

Variant **get_item_metadata**(idx: int) const

Retrieves the metadata of an item. Metadata may be any type and can be used to store extra information about an item, such as an external string ID.

String **get_item_text**(idx: int) const

Returns the text of the item at index `idx`.

String **get_item_tooltip**(idx: int) const

Returns the tooltip of the item at index `idx`.

PopupMenu **get_popup**() const

Returns the PopupMenu contained in this button.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their Window.visible property.

---

int **get_selectable_item**(from_last: bool = false) const

Returns the index of the first item which is not disabled, or marked as a separator. If `from_last` is `true`, the items will be searched in reverse order.

Returns `-1` if no item is found.

---

int **get_selected_id**() const

Returns the ID of the selected item, or `-1` if no item is selected.

Variant **get_selected_metadata**() const

Gets the metadata of the selected item. Metadata for items can be set using set_item_metadata.

bool **has_selectable_items**() const 🔗

Returns `true` if this button contains at least one item which is not disabled, or marked as a separator.

bool **is_item_disabled**(idx: int) const 🔗

Returns `true` if the item at index `idx` is disabled.

bool **is_item_separator**(idx: int) const 🔗

Returns `true` if the item at index `idx` is marked as a separator.

void **remove_item**(idx: int) 🔗

Removes the item at index `idx`.

void **select**(idx: int) 🔗

Selects an item by index and makes it the current item. This will work even if the item is disabled.

Passing `-1` as the index deselects any currently selected item.

void **set_disable_shortcuts**(disabled: bool) 🔗

If `true`, shortcuts are disabled and cannot be used to trigger the button.

---

void **set_item_disabled**(idx: int, disabled: bool)

Sets whether the item at index `idx` is disabled.

Disabled items are drawn differently in the dropdown and are not selectable by the user. If the current selected item is set as disabled, it will remain selected.

---

void **set_item_icon**(idx: int, texture: Texture2D)

Sets the icon of the item at index `idx`.

---

void **set_item_id**(idx: int, id: int)

Sets the ID of the item at index `idx`.

---

void **set_item_metadata**(idx: int, metadata: Variant)

Sets the metadata of an item. Metadata may be of any type and can be used to store extra information about an item, such as an external string ID.

---

void **set_item_text**(idx: int, text: String)

Sets the text of the item at index `idx`.

---

void **set_item_tooltip**(idx: int, tooltip: String)

Sets the tooltip of the item at index `idx`.

---

void **show_popup**()

Adjusts popup position and sizing for the **OptionButton**, then shows the PopupMenu. Prefer this over using `get_popup().popup()`.

Theme Property Descriptions

int **arrow_margin** = `4`

The horizontal space between the arrow icon and the right edge of the button.

---

int **modulate_arrow** = `0`

If different than `0`, the arrow icon will be modulated to the font color.

Texture2D **arrow**

The arrow icon to be drawn on the right end of the button.

Panel

Inherits: Control **<** CanvasItem **<** Node **<** Object

A GUI control that displays a StyleBox.

Description

Panel is a GUI control that displays a StyleBox. See also PanelContainer.

Tutorials

- 2D Role Playing Game (RPG) Demo [https://godotengine.org/asset-library/asset/2729]
- Hierarchical Finite State Machine Demo [https://godotengine.org/asset-library/asset/2714]

Theme Properties

StyleBox	panel

Theme Property Descriptions

StyleBox **panel** 🔗

The StyleBox of this control.

PanelContainer

Inherits: Container **<** Control **<** CanvasItem **<** Node **<** Object

Inherited By: ScriptEditor

A container that keeps its child controls within the area of a StyleBox.

Description

A container that keeps its child controls within the area of a StyleBox. Useful for giving controls an outline.

Tutorials

- Using Containers
- 2D Role Playing Game (RPG) Demo [https://godotengine.org/asset-library/asset/2729]

Properties

MouseFilter	mouse_filter	`0` (overrides Control)

Theme Properties

StyleBox	panel

Theme Property Descriptions

StyleBox **panel** 🔗

The style of **PanelContainer**'s background.

Parallax2D

Experimental: This node is meant to replace ParallaxBackground and ParallaxLayer. The implementation may change in the future.

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A node used to create a parallax scrolling background.

Description

A **Parallax2D** is used to create a parallax effect. It can move at a different speed relative to the camera movement using scroll_scale. This creates an illusion of depth in a 2D game. If manual scrolling is desired, the Camera2D position can be ignored with ignore_camera_scroll.

Note: Any changes to this node's position made after it enters the scene tree will be overridden if ignore_camera_scroll is `false` or screen_offset is modified.

Tutorials

- 2D Parallax

Properties

Vector2	autoscroll	`Vector2(0, 0)`
bool	follow_viewport	`true`

bool	ignore_camera_scroll	`false`
Vector2	limit_begin	`Vector2(-1e+07, -1e+07)`
Vector2	limit_end	`Vector2(1e+07, 1e+07)`
PhysicsInterpolationMode	physics_interpolation_mode	`2` (overrides Node)
Vector2	repeat_size	`Vector2(0, 0)`
int	repeat_times	`1`
Vector2	screen_offset	`Vector2(0, 0)`
Vector2	scroll_offset	`Vector2(0, 0)`
Vector2	scroll_scale	`Vector2(1, 1)`

Property Descriptions

Vector2 **autoscroll** = `Vector2(0, 0)` 🔗

- void **set_autoscroll**(value: Vector2)
- Vector2 **get_autoscroll**()

Velocity at which the offset scrolls automatically, in pixels per second.

---

bool **follow_viewport** = `true` 🔗

- void **set_follow_viewport**(value: bool)
- bool **get_follow_viewport**()

If `true`, this **Parallax2D** is offset by the current camera's position. If the **Parallax2D** is in a CanvasLayer separate from the current camera, it may be desired to match the value with CanvasLayer.follow_viewport_enabled.

---

bool **ignore_camera_scroll** = `false` 🔗

- void **set_ignore_camera_scroll**(value: bool)
- bool **is_ignore_camera_scroll**()

If `true`, **Parallax2D**'s position is not affected by the position of the camera.

Vector2 **limit_begin** = `Vector2(-1e+07, -1e+07)` 🔗

- void **set_limit_begin**(value: Vector2)
- Vector2 **get_limit_begin**()

Top-left limits for scrolling to begin. If the camera is outside of this limit, the **Parallax2D** stops scrolling. Must be lower than limit_end minus the viewport size to work.

Vector2 **limit_end** = `Vector2(1e+07, 1e+07)` 🔗

- void **set_limit_end**(value: Vector2)
- Vector2 **get_limit_end**()

Bottom-right limits for scrolling to end. If the camera is outside of this limit, the **Parallax2D** will stop scrolling. Must be higher than limit_begin and the viewport size combined to work.

Vector2 **repeat_size** = `Vector2(0, 0)`

- void **set_repeat_size**(value: Vector2)
- Vector2 **get_repeat_size**()

Repeats the Texture2D of each of this node's children and offsets them by this value. When scrolling, the node's position loops, giving the illusion of an infinite scrolling background if the values are larger than the screen size. If an axis is set to `0`, the Texture2D will not be repeated.

---

int **repeat_times** = `1`

- void **set_repeat_times**(value: int)
- int **get_repeat_times**()

Overrides the amount of times the texture repeats. Each texture copy spreads evenly from the original by repeat_size. Useful for when zooming out with a camera.

Vector2 **screen_offset** = `Vector2(0, 0)`

- void **set_screen_offset**(value: Vector2)
- Vector2 **get_screen_offset**()

Offset used to scroll this **Parallax2D**. This value is updated automatically unless ignore_camera_scroll is `true`.

Vector2 **scroll_offset** = `Vector2(0, 0)`

- void **set_scroll_offset**(value: Vector2)
- Vector2 **get_scroll_offset**()

The **Parallax2D**'s offset. Similar to screen_offset and Node2D.position, but will not be overridden.

Note: Values will loop if repeat_size is set higher than `0`.

Vector2 **scroll_scale** = `Vector2(1, 1)`

- void **set_scroll_scale**(value: Vector2)
- Vector2 **get_scroll_scale**()

Multiplier to the final **Parallax2D**'s offset. Can be used to simulate distance from the camera.

For example, a value of `1` scrolls at the same speed as the camera. A value greater than `1` scrolls faster, making objects appear closer. Less than `1` scrolls slower, making objects appear further, and a value of `0` stops the objects completely.

ParallaxBackground

Inherits: CanvasLayer **<** Node **<** Object

A node used to create a parallax scrolling background.

Description

A ParallaxBackground uses one or more ParallaxLayer child nodes to create a parallax effect. Each ParallaxLayer can move at a different speed using ParallaxLayer.motion_offset. This creates an illusion of depth in a 2D game. If not used with a Camera2D, you must manually calculate the scroll_offset.

Note: Each **ParallaxBackground** is drawn on one specific Viewport and cannot be shared between multiple Viewports, see CanvasLayer.custom_viewport. When using multiple Viewports, for example in a split-screen game, you need create an individual **ParallaxBackground** for each Viewport you want it to be drawn on.

Properties

int	layer	`-100` (overrides CanvasLayer)
Vector2	scroll_base_offset	`Vector2(0, 0)`
Vector2	scroll_base_scale	`Vector2(1, 1)`

bool	scroll_ignore_camera_zoom	`false`
Vector2	scroll_limit_begin	`Vector2(0, 0)`
Vector2	scroll_limit_end	`Vector2(0, 0)`
Vector2	scroll_offset	`Vector2(0, 0)`

Property Descriptions

Vector2 **scroll_base_offset** = `Vector2(0, 0)`

- void **set_scroll_base_offset**(value: Vector2)
- Vector2 **get_scroll_base_offset**()

The base position offset for all ParallaxLayer children.

Vector2 **scroll_base_scale** = `Vector2(1, 1)`

- void **set_scroll_base_scale**(value: Vector2)
- Vector2 **get_scroll_base_scale**()

The base motion scale for all ParallaxLayer children.

bool **scroll_ignore_camera_zoom** = `false`

- void **set_ignore_camera_zoom**(value: bool)
- bool **is_ignore_camera_zoom**()

If `true`, elements in ParallaxLayer child aren't affected by the zoom level of the camera.

Vector2 **scroll_limit_begin** = `Vector2(0, 0)`

- void **set_limit_begin**(value: Vector2)
- Vector2 **get_limit_begin**()

Top-left limits for scrolling to begin. If the camera is outside of this limit, the background will stop scrolling. Must be lower than scroll_limit_end to work.

Vector2 **scroll_limit_end** = `Vector2(0, 0)`

- void **set_limit_end**(value: Vector2)
- Vector2 **get_limit_end**()

Bottom-right limits for scrolling to end. If the camera is outside of this limit, the background will stop scrolling. Must be higher than scroll_limit_begin to work.

Vector2 **scroll_offset** = `Vector2(0, 0)`

- void **set_scroll_offset**(value: Vector2)
- Vector2 **get_scroll_offset**()

The ParallaxBackground's scroll value. Calculated automatically when using a Camera2D, but can be used to manually manage scrolling when no camera is present.

ParallaxLayer

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A parallax scrolling layer to be used with ParallaxBackground.

Description

A ParallaxLayer must be the child of a ParallaxBackground node. Each ParallaxLayer can be set to move at different speeds relative to the camera movement or the ParallaxBackground.scroll_offset value.

This node's children will be affected by its scroll offset.

Note: Any changes to this node's position and scale made after it enters the scene will be ignored.

Properties

Vector2	motion_mirroring	`Vector2(0, 0)`
Vector2	motion_offset	`Vector2(0, 0)`
Vector2	motion_scale	`Vector2(1, 1)`

PhysicsInterpolationMode	physics_interpolation_mode	`2` (overrides Node)

Property Descriptions

Vector2 **motion_mirroring** = `Vector2(0, 0)` 🔗

- void **set_mirroring**(value: Vector2)
- Vector2 **get_mirroring**()

The interval, in pixels, at which the **ParallaxLayer** is drawn repeatedly. Useful for creating an infinitely scrolling background. If an axis is set to `0`, the **ParallaxLayer** will be drawn only once along that direction.

Note: If you want the repetition to pixel-perfect match a Texture2D displayed by a child node, you should account for any scale applied to the texture when defining this interval. For example, if you use a child Sprite2D scaled to `0.5` to display a 600x600 texture, and want this sprite to be repeated continuously horizontally, you should set the mirroring to `Vector2(300, 0)`.

Note: If the length of the viewport axis is bigger than twice the repeated axis size, it will not repeat infinitely, as the parallax layer only draws 2 instances of the layer at any given time. The visibility window is calculated from the parent ParallaxBackground's position, not the layer's own position. So, if you use mirroring, **do not** change the **ParallaxLayer** position relative to its parent. Instead, if you need to adjust the background's position, set the CanvasLayer.offset property in the parent ParallaxBackground.

Note: Despite the name, the layer will not be mirrored, it will only be repeated.

Vector2 **motion_offset** = `Vector2(0, 0)`

- void **set_motion_offset**(value: Vector2)
- Vector2 **get_motion_offset**()

The ParallaxLayer's offset relative to the parent ParallaxBackground's ParallaxBackground.scroll_offset.

Vector2 **motion_scale** = `Vector2(1, 1)`

- void **set_motion_scale**(value: Vector2)
- Vector2 **get_motion_scale**()

Multiplies the ParallaxLayer's motion. If an axis is set to `0`, it will not scroll.

Path2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Contains a Curve2D path for PathFollow2D nodes to follow.

Description

Can have PathFollow2D child nodes moving along the Curve2D. See PathFollow2D for more information on usage.

Note: The path is considered as relative to the moved nodes (children of PathFollow2D). As such, the curve should usually start with a zero vector (`(0, 0)`).

Properties

Curve2D	curve

Property Descriptions

Curve2D **curve** 🔗

- void **set_curve**(value: Curve2D)
- Curve2D **get_curve**()

A Curve2D describing the path.

Path3D

Inherits: Node3D < Node < Object

Contains a Curve3D path for PathFollow3D nodes to follow.

Description

Can have PathFollow3D child nodes moving along the Curve3D. See PathFollow3D for more information on the usage.

Note that the path is considered as relative to the moved nodes (children of PathFollow3D). As such, the curve should usually start with a zero vector `(0, 0, 0)`.

Properties

Curve3D	curve

Signals

curve_changed() 🔗

Emitted when the curve changes.

Property Descriptions

Curve3D **curve** 🔗

- void **set_curve**(value: Curve3D)
- Curve3D **get_curve**()

A Curve3D describing the path.

PathFollow2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Point sampler for a Path2D.

Description

This node takes its parent Path2D, and returns the coordinates of a point within it, given a distance from the first vertex.

It is useful for making other nodes follow a path, without coding the movement pattern. For that, the nodes must be children of this node. The descendant nodes will then move accordingly when setting the progress in this node.

Properties

bool	cubic_interp	`true`
float	h_offset	`0.0`
bool	loop	`true`
float	progress	`0.0`
float	progress_ratio	`0.0`

bool	rotates	`true`
float	v_offset	`0.0`

Property Descriptions

bool **cubic_interp** = `true`

- void **set_cubic_interpolation**(value: bool)
- bool **get_cubic_interpolation**()

If `true`, the position between two cached points is interpolated cubically, and linearly otherwise.

The points along the Curve2D of the Path2D are precomputed before use, for faster calculations. The point at the requested offset is then calculated interpolating between two adjacent cached points. This may present a problem if the curve makes sharp turns, as the cached points may not follow the curve closely enough.

There are two answers to this problem: either increase the number of cached points and increase memory consumption, or make a cubic interpolation between two points at the cost of (slightly) slower calculations.

---

float **h_offset** = `0.0`

- void **set_h_offset**(value: float)
- float **get_h_offset**()

The node's offset along the curve.

---

bool **loop** = `true` 🔗

- void **set_loop**(value: bool)
- bool **has_loop**()

If `true`, any offset outside the path's length will wrap around, instead of stopping at the ends. Use it for cyclic paths.

---

float **progress** = `0.0` 🔗

- void **set_progress**(value: float)
- float **get_progress**()

The distance along the path, in pixels. Changing this value sets this node's position to a point within the path.

---

float **progress_ratio** = `0.0` 🔗

- void **set_progress_ratio**(value: float)
- float **get_progress_ratio**()

The distance along the path as a number in the range 0.0 (for the first vertex) to 1.0 (for the last). This is just another way of expressing the progress within the path, as the offset supplied is multiplied internally by the path's length.

---

bool **rotates** = `true` 🔗

- void **set_rotates**(value: bool)
- bool **is_rotating**()

If `true`, this node rotates to follow the path, with the +X direction facing forward on the path.

---

float **v_offset** = `0.0` 🔗

- void **set_v_offset**(value: float)
- float **get_v_offset**()

The node's offset perpendicular to the curve.

PathFollow3D

Inherits: Node3D **<** Node **<** Object

Point sampler for a Path3D.

Description

This node takes its parent Path3D, and returns the coordinates of a point within it, given a distance from the first vertex.

It is useful for making other nodes follow a path, without coding the movement pattern. For that, the nodes must be children of this node. The descendant nodes will then move accordingly when setting the progress in this node.

Properties

bool	cubic_interp	`true`
float	h_offset	`0.0`
bool	loop	`true`
float	progress	`0.0`
float	progress_ratio	`0.0`

RotationMode	rotation_mode	`3`
bool	tilt_enabled	`true`
bool	use_model_front	`false`
float	v_offset	`0.0`

Methods

Transform3D	correct_posture(transform: Transform3D, rotation_mode: RotationMode) static

---

Enumerations

enum **RotationMode**: 🔗

RotationMode **ROTATION_NONE** = `0`

Forbids the PathFollow3D to rotate.

RotationMode **ROTATION_Y** = `1`

Allows the PathFollow3D to rotate in the Y axis only.

RotationMode **ROTATION_XY** = `2`

Allows the PathFollow3D to rotate in both the X, and Y axes.

RotationMode **ROTATION_XYZ** = `3`

Allows the PathFollow3D to rotate in any axis.

RotationMode **ROTATION_ORIENTED** = `4`

Uses the up vector information in a Curve3D to enforce orientation. This rotation mode requires the Path3D's Curve3D.up_vector_enabled property to be set to `true`.

Property Descriptions

bool **cubic_interp** = `true` 🔗

- void **set_cubic_interpolation**(value: bool)
- bool **get_cubic_interpolation**()

If `true`, the position between two cached points is interpolated cubically, and linearly otherwise.

The points along the Curve3D of the Path3D are precomputed before use, for faster calculations. The point at the requested offset is then calculated interpolating between two adjacent cached points. This may present a problem if the curve makes sharp turns, as the cached points may not follow the curve closely enough.

There are two answers to this problem: either increase the number of cached points and increase memory consumption, or make a cubic interpolation between two points at the cost of (slightly) slower calculations.

---

float **h_offset** = `0.0` 🔗

- void **set_h_offset**(value: float)

- float **get_h_offset**()

The node's offset along the curve.

---

bool **loop** = `true` 🔗

- void **set_loop**(value: bool)
- bool **has_loop**()

If `true`, any offset outside the path's length will wrap around, instead of stopping at the ends. Use it for cyclic paths.

---

float **progress** = `0.0` 🔗

- void **set_progress**(value: float)
- float **get_progress**()

The distance from the first vertex, measured in 3D units along the path. Changing this value sets this node's position to a point within the path.

---

float **progress_ratio** = `0.0` 🔗

- void **set_progress_ratio**(value: float)
- float **get_progress_ratio**()

The distance from the first vertex, considering 0.0 as the first vertex and 1.0 as the last. This is just another way of expressing the progress within the path, as the progress supplied is multiplied internally by the path's length.

RotationMode **rotation_mode** = `3` 🔗

- void **set_rotation_mode**(value: RotationMode)
- RotationMode **get_rotation_mode**()

Allows or forbids rotation on one or more axes, depending on the RotationMode constants being used.

---

bool **tilt_enabled** = `true` 🔗

- void **set_tilt_enabled**(value: bool)
- bool **is_tilt_enabled**()

If `true`, the tilt property of Curve3D takes effect.

---

bool **use_model_front** = `false` 🔗

- void **set_use_model_front**(value: bool)
- bool **is_using_model_front**()

If `true`, the node moves on the travel path with orienting the +Z axis as forward. See also Vector3.FORWARD and Vector3.MODEL_FRONT.

---

float **v_offset** = `0.0` 🔗

- void **set_v_offset**(value: float)
- float **get_v_offset**()

The node's offset perpendicular to the curve.

Method Descriptions

Transform3D **correct_posture**(transform: Transform3D, rotation_mode: RotationMode) static 🔗

Correct the `transform. rotation_mode` implicitly specifies how posture (forward, up and sideway direction) is calculated.

PhysicalBone2D

Inherits: RigidBody2D **<** PhysicsBody2D **<** CollisionObject2D **<** Node2D **<** CanvasItem **<** Node **<** Object

A RigidBody2D-derived node used to make Bone2Ds in a Skeleton2D react to physics.

Description

The **PhysicalBone2D** node is a RigidBody2D-based node that can be used to make Bone2Ds in a Skeleton2D react to physics.

Note: To make the Bone2Ds visually follow the **PhysicalBone2D** node, use a SkeletonModification2DPhysicalBones modification on the Skeleton2D parent.

Note: The **PhysicalBone2D** node does not automatically create a Joint2D node to keep **PhysicalBone2D** nodes together. They must be created manually. For most cases, you want to use a PinJoint2D node. The **PhysicalBone2D** node will automatically configure the Joint2D node once it's been added as a child node.

Properties

bool	auto_configure_joint	`true`
int	bone2d_index	`-1`

NodePath	bone2d_nodepath	`NodePath("")`
bool	follow_bone_when_simulating	`false`
bool	simulate_physics	`false`

Methods

Joint2D	get_joint() const
bool	is_simulating_physics() const

Property Descriptions

bool **auto_configure_joint** = `true`

- void **set_auto_configure_joint**(value: bool)
- bool **get_auto_configure_joint**()

If `true`, the **PhysicalBone2D** will automatically configure the first Joint2D child node. The automatic configuration is limited to setting up the node properties and positioning the Joint2D.

---

int **bone2d_index** = `-1`

- void **set_bone2d_index**(value: int)

- int **get_bone2d_index**()

The index of the Bone2D that this **PhysicalBone2D** should simulate.

NodePath **bone2d_nodepath** = `NodePath("")`

- void **set_bone2d_nodepath**(value: NodePath)
- NodePath **get_bone2d_nodepath**()

The NodePath to the Bone2D that this **PhysicalBone2D** should simulate.

bool **follow_bone_when_simulating** = `false`

- void **set_follow_bone_when_simulating**(value: bool)
- bool **get_follow_bone_when_simulating**()

If `true`, the **PhysicalBone2D** will keep the transform of the bone it is bound to when simulating physics.

bool **simulate_physics** = `false`

- void **set_simulate_physics**(value: bool)
- bool **get_simulate_physics**()

If `true`, the **PhysicalBone2D** will start simulating using physics. If `false`, the **PhysicalBone2D** will follow the transform of the Bone2D node.

Note: To have the Bone2Ds visually follow the **PhysicalBone2D**, use a SkeletonModification2DPhysicalBones modification on the Skeleton2D node with the Bone2D nodes.

Method Descriptions

Joint2D **get_joint**() const 🔗

Returns the first Joint2D child node, if one exists. This is mainly a helper function to make it easier to get the Joint2D that the **PhysicalBone2D** is autoconfiguring.

---

bool **is_simulating_physics**() const 🔗

Returns a boolean that indicates whether the **PhysicalBone2D** is running and simulating using the Godot 2D physics engine. When `true`, the PhysicalBone2D node is using physics.

PhysicalBone3D

Inherits: PhysicsBody3D **<** CollisionObject3D **<** Node3D **<** Node **<** Object

A physics body used to make bones in a Skeleton3D react to physics.

Description

The **PhysicalBone3D** node is a physics body that can be used to make bones in a Skeleton3D react to physics.

Note: In order to detect physical bones with raycasts, the SkeletonModifier3D.active property of the parent PhysicalBoneSimulator3D must be `true` and the Skeleton3D's bone must be assigned to **PhysicalBone3D** correctly; it means that get_bone_id should return a valid id (`>= 0`).

Properties

float	angular_damp	`0.0`
DampMode	angular_damp_mode	`0`
Vector3	angular_velocity	`Vector3(0, 0, 0)`
Transform3D	body_offset	`Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)`

float	bounce	`0.0`
bool	can_sleep	`true`
bool	custom_integrator	`false`
float	friction	`1.0`
float	gravity_scale	`1.0`
Transform3D	joint_offset	`Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)`
Vector3	joint_rotation	`Vector3(0, 0, 0)`
JointType	joint_type	`0`
float	linear_damp	`0.0`
DampMode	linear_damp_mode	`0`
Vector3	linear_velocity	`Vector3(0, 0, 0)`
float	mass	`1.0`

Methods

void	_integrate_forces(state: PhysicsDirectBodyState3D) virtual
void	apply_central_impulse(impulse: Vector3)
void	apply_impulse(impulse: Vector3, position: Vector3 = Vector3(0, 0, 0))
int	get_bone_id() const
bool	get_simulate_physics()
bool	is_simulating_physics()

Enumerations

enum **DampMode**: 🔗

DampMode **DAMP_MODE_COMBINE** = `0`

In this mode, the body's damping value is added to any value set in areas or the default value.

DampMode **DAMP_MODE_REPLACE** = `1`

In this mode, the body's damping value replaces any value set in areas or the default value.

---

enum **JointType**: 🔗

JointType **JOINT_TYPE_NONE** = `0`

There is currently no description for this enum. Please help us by contributing one!

JointType **JOINT_TYPE_PIN** = `1`

There is currently no description for this enum. Please help us by contributing one!

JointType **JOINT_TYPE_CONE** = `2`

There is currently no description for this enum. Please help us by contributing one!

JointType **JOINT_TYPE_HINGE** = `3`

There is currently no description for this enum. Please help us by contributing one!

JointType **JOINT_TYPE_SLIDER** = `4`

There is currently no description for this enum. Please help us by contributing one!

JointType **JOINT_TYPE_6DOF** = `5`

There is currently no description for this enum. Please help us by contributing one!

Property Descriptions

float **angular_damp** = `0.0` 🔗

- void **set_angular_damp**(value: float)
- float **get_angular_damp**()

Damps the body's rotation. By default, the body will use the **Default Angular Damp** in **Project > Project Settings > Physics > 3d** or any value override set by an Area3D the body is in. Depending on angular_damp_mode, you can set angular_damp to be added to or to replace the body's damping value.

See ProjectSettings.physics/3d/default_angular_damp for more details about damping.

DampMode **angular_damp_mode** = `0` 🔗

- void **set_angular_damp_mode**(value: DampMode)
- DampMode **get_angular_damp_mode**()

Defines how angular_damp is applied. See DampMode for possible values.

Vector3 **angular_velocity** = `Vector3(0, 0, 0)` 🔗

- void **set_angular_velocity**(value: Vector3)
- Vector3 **get_angular_velocity**()

The PhysicalBone3D's rotational velocity in *radians* per second.

Transform3D **body_offset** = `Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)` 🔗

- void **set_body_offset**(value: Transform3D)
- Transform3D **get_body_offset**()

Sets the body's transform.

---

float **bounce** = `0.0`

- void **set_bounce**(value: float)
- float **get_bounce**()

The body's bounciness. Values range from `0` (no bounce) to `1` (full bounciness).

Note: Even with bounce set to `1.0`, some energy will be lost over time due to linear and angular damping. To have a **PhysicalBone3D** that preserves all its energy over time, set bounce to `1.0`, linear_damp_mode to DAMP_MODE_REPLACE, linear_damp to `0.0`, angular_damp_mode to DAMP_MODE_REPLACE, and angular_damp to `0.0`.

---

bool **can_sleep** = `true`

- void **set_can_sleep**(value: bool)
- bool **is_able_to_sleep**()

If `true`, the body is deactivated when there is no movement, so it will not take part in the simulation until it is awakened by an external force.

---

bool **custom_integrator** = `false`

- void **set_use_custom_integrator**(value: bool)
- bool **is_using_custom_integrator**()

If `true`, the standard force integration (like gravity or damping) will be disabled for this body. Other than collision response, the body will only move as determined by the _integrate_forces method, if that virtual method is overridden.

Setting this property will call the method PhysicsServer3D.body_set_omit_force_integration internally.

float **friction** = `1.0`

- void **set_friction**(value: float)
- float **get_friction**()

The body's friction, from `0` (frictionless) to `1` (max friction).

float **gravity_scale** = `1.0`

- void **set_gravity_scale**(value: float)
- float **get_gravity_scale**()

This is multiplied by the global 3D gravity setting found in **Project > Project Settings > Physics > 3d** to produce the body's gravity. For example, a value of 1 will be normal gravity, 2 will apply double gravity, and 0.5 will apply half gravity to this object.

Transform3D **joint_offset** = `Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)`

- void **set_joint_offset**(value: Transform3D)
- Transform3D **get_joint_offset**()

Sets the joint's transform.

Vector3 **joint_rotation** = `Vector3(0, 0, 0)`

- void **set_joint_rotation**(value: Vector3)
- Vector3 **get_joint_rotation**()

Sets the joint's rotation in radians.

JointType **joint_type** = `0` 🔗

- void **set_joint_type**(value: JointType)
- JointType **get_joint_type**()

Sets the joint type. See JointType for possible values.

float **linear_damp** = `0.0` 🔗

- void **set_linear_damp**(value: float)
- float **get_linear_damp**()

Damps the body's movement. By default, the body will use the **Default Linear Damp** in **Project > Project Settings > Physics > 3d** or any value override set by an Area3D the body is in. Depending on linear_damp_mode, you can set linear_damp to be added to or to replace the body's damping value.

See ProjectSettings.physics/3d/default_linear_damp for more details about damping.

DampMode **linear_damp_mode** = `0` 🔗

- void **set_linear_damp_mode**(value: DampMode)
- DampMode **get_linear_damp_mode**()

Defines how linear_damp is applied. See DampMode for possible values.

Vector3 **linear_velocity** = `Vector3(0, 0, 0)` 🔗

- void **set_linear_velocity**(value: Vector3)
- Vector3 **get_linear_velocity**()

The body's linear velocity in units per second. Can be used sporadically, but **don't set this every frame**, because physics may run in another thread and runs at a different granularity. Use _integrate_forces as your process loop for precise control of the body state.

---

float **mass** = `1.0` 🔗

- void **set_mass**(value: float)
- float **get_mass**()

The body's mass.

Method Descriptions

void **_integrate_forces**(state: PhysicsDirectBodyState3D) virtual 🔗

Called during physics processing, allowing you to read and safely modify the simulation state for the object. By default, it is called before the standard force integration, but the custom_integrator property allows you to disable the standard force integration and do fully custom force integration for a body.

---

void **apply_central_impulse**(impulse: Vector3) 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **apply_impulse**(impulse: Vector3, position: Vector3 = Vector3(0, 0, 0))

There is currently no description for this method. Please help us by contributing one!

---

int **get_bone_id**() const

There is currently no description for this method. Please help us by contributing one!

---

bool **get_simulate_physics**()

There is currently no description for this method. Please help us by contributing one!

---

bool **is_simulating_physics**()

There is currently no description for this method. Please help us by contributing one!

PhysicalBoneSimulator3D

Inherits: SkeletonModifier3D **<** Node3D **<** Node **<** Object

Node that can be the parent of PhysicalBone3D and can apply the simulation results to Skeleton3D.

Description

Node that can be the parent of PhysicalBone3D and can apply the simulation results to Skeleton3D.

Methods

bool	is_simulating_physics() const
void	physical_bones_add_collision_exception(exception: RID)
void	physical_bones_remove_collision_exception(exception: RID)
void	physical_bones_start_simulation(bones: Array[StringName] = [])
void	physical_bones_stop_simulation()

Method Descriptions

bool **is_simulating_physics**() const

Returns a boolean that indicates whether the **PhysicalBoneSimulator3D** is running and simulating.

---

void **physical_bones_add_collision_exception**(exception: RID)

Adds a collision exception to the physical bone.

Works just like the RigidBody3D node.

---

void **physical_bones_remove_collision_exception**(exception: RID)

Removes a collision exception to the physical bone.

Works just like the RigidBody3D node.

---

void **physical_bones_start_simulation**(bones: Array[StringName] = [])

Tells the PhysicalBone3D nodes in the Skeleton to start simulating and reacting to the physics world.

Optionally, a list of bone names can be passed-in, allowing only the passed-in bones to be simulated.

---

void **physical_bones_stop_simulation**()

Tells the PhysicalBone3D nodes in the Skeleton to stop simulating.

PhysicsBody2D

Inherits: CollisionObject2D **<** Node2D **<** CanvasItem **<** Node **<** Object

Inherited By: CharacterBody2D, RigidBody2D, StaticBody2D

Abstract base class for 2D game objects affected by physics.

Description

PhysicsBody2D is an abstract base class for 2D game objects affected by physics. All 2D physics bodies inherit from it.

Tutorials

- Physics introduction

Properties

bool	input_pickable	`false` (overrides CollisionObject2D)

Methods

void	add_collision_exception_with(body: Node)

Array[PhysicsBody2D]	get_collision_exceptions()
Vector2	get_gravity() const
KinematicCollision2D	move_and_collide(motion: Vector2, test_only: bool = false, safe_margin: float = 0.08, recovery_as_collision: bool = false)
void	remove_collision_exception_with(body: Node)
bool	test_move(from: Transform2D, motion: Vector2, collision: KinematicCollision2D = null, safe_margin: float = 0.08, recovery_as_collision: bool = false)

Method Descriptions

void **add_collision_exception_with**(body: Node) 🔗

Adds a body to the list of bodies that this body can't collide with.

Array[PhysicsBody2D] **get_collision_exceptions**() 🔗

Returns an array of nodes that were added as collision exceptions for this body.

Vector2 **get_gravity**() const 🔗

Returns the gravity vector computed from all sources that can affect the body, including all gravity overrides from Area2D nodes and the global world gravity.

KinematicCollision2D **move_and_collide**(motion: Vector2, test_only: bool = false, safe_margin: float = 0.08, recovery_as_collision: bool = false) 🔗

Moves the body along the vector `motion`. In order to be frame rate independent in Node._physics_process or Node._process, `motion` should be computed using `delta`.

Returns a KinematicCollision2D, which contains information about the collision when stopped, or when touching another body along the motion.

If `test_only` is `true`, the body does not move but the would-be collision information is given.

`safe_margin` is the extra margin used for collision recovery (see CharacterBody2D.safe_margin for more details).

If `recovery_as_collision` is `true`, any depenetration from the recovery phase is also reported as a collision; this is used e.g. by CharacterBody2D for improving floor detection during floor snapping.

void **remove_collision_exception_with**(body: Node) 🔗

Removes a body from the list of bodies that this body can't collide with.

bool **test_move**(from: Transform2D, motion: Vector2, collision: KinematicCollision2D = null, safe_margin: float = 0.08, recovery_as_collision: bool = false)

Checks for collisions without moving the body. In order to be frame rate independent in Node._physics_process or Node._process, `motion` should be computed using `delta`.

Virtually sets the node's position, scale and rotation to that of the given Transform2D, then tries to move the body along the vector `motion`. Returns `true` if a collision would stop the body from moving along the whole path.

`collision` is an optional object of type KinematicCollision2D, which contains additional information about the collision when stopped, or when touching another body along the motion.

`safe_margin` is the extra margin used for collision recovery (see CharacterBody2D.safe_margin for more details).

If `recovery_as_collision` is `true`, any depenetration from the recovery phase is also reported as a collision; this is useful for checking whether the body would *touch* any other bodies.

PhysicsBody3D

Inherits: CollisionObject3D **<** Node3D **<** Node **<** Object

Inherited By: CharacterBody3D, PhysicalBone3D, RigidBody3D, StaticBody3D

Abstract base class for 3D game objects affected by physics.

Description

PhysicsBody3D is an abstract base class for 3D game objects affected by physics. All 3D physics bodies inherit from it.

Warning: With a non-uniform scale, this node will likely not behave as expected. It is advised to keep its scale the same on all axes and adjust its collision shape(s) instead.

Tutorials

- Physics introduction

Properties

bool	axis_lock_angular_x	`false`
bool	axis_lock_angular_y	`false`

bool	axis_lock_angular_z	`false`
bool	axis_lock_linear_x	`false`
bool	axis_lock_linear_y	`false`
bool	axis_lock_linear_z	`false`

Methods

void	add_collision_exception_with(body: Node)
bool	get_axis_lock(axis: BodyAxis) const
Array[PhysicsBody3D]	get_collision_exceptions()
Vector3	get_gravity() const
KinematicCollision3D	move_and_collide(motion: Vector3, test_only: bool = false, safe_margin: float = 0.001, recovery_as_collision: bool = false, max_collisions: int = 1)
void	remove_collision_exception_with(body:

	Node)
void	set_axis_lock(axis: BodyAxis, lock: bool)
bool	test_move(from: Transform3D, motion: Vector3, collision: KinematicCollision3D = null, safe_margin: float = 0.001, recovery_as_collision: bool = false, max_collisions: int = 1)

Property Descriptions

bool **axis_lock_angular_x** = `false`

- void **set_axis_lock**(axis: BodyAxis, lock: bool)
- bool **get_axis_lock**(axis: BodyAxis) const

Lock the body's rotation in the X axis.

bool **axis_lock_angular_y** = `false`

- void **set_axis_lock**(axis: BodyAxis, lock: bool)
- bool **get_axis_lock**(axis: BodyAxis) const

Lock the body's rotation in the Y axis.

bool **axis_lock_angular_z** = `false`

- void **set_axis_lock**(axis: BodyAxis, lock: bool)
- bool **get_axis_lock**(axis: BodyAxis) const

Lock the body's rotation in the Z axis.

---

bool **axis_lock_linear_x** = `false` 🔗

- void **set_axis_lock**(axis: BodyAxis, lock: bool)
- bool **get_axis_lock**(axis: BodyAxis) const

Lock the body's linear movement in the X axis.

---

bool **axis_lock_linear_y** = `false` 🔗

- void **set_axis_lock**(axis: BodyAxis, lock: bool)
- bool **get_axis_lock**(axis: BodyAxis) const

Lock the body's linear movement in the Y axis.

---

bool **axis_lock_linear_z** = `false` 🔗

- void **set_axis_lock**(axis: BodyAxis, lock: bool)
- bool **get_axis_lock**(axis: BodyAxis) const

Lock the body's linear movement in the Z axis.

Method Descriptions

void **add_collision_exception_with**(body: Node) 🔗

Adds a body to the list of bodies that this body can't collide with.

---

bool **get_axis_lock**(axis: BodyAxis) const 🔗

Returns `true` if the specified linear or rotational `axis` is locked.

Array[PhysicsBody3D] **get_collision_exceptions**()

Returns an array of nodes that were added as collision exceptions for this body.

Vector3 **get_gravity**() const

Returns the gravity vector computed from all sources that can affect the body, including all gravity overrides from Area3D nodes and the global world gravity.

KinematicCollision3D **move_and_collide**(motion: Vector3, test_only: bool = false, safe_margin: float = 0.001, recovery_as_collision: bool = false, max_collisions: int = 1)

Moves the body along the vector `motion`. In order to be frame rate independent in Node._physics_process or Node._process, `motion` should be computed using `delta`.

The body will stop if it collides. Returns a KinematicCollision3D, which contains information about the collision when stopped, or when touching another body along the motion.

If `test_only` is `true`, the body does not move but the would-be collision information is given.

`safe_margin` is the extra margin used for collision recovery (see CharacterBody3D.safe_margin for more details).

If `recovery_as_collision` is `true`, any depenetration from the recovery phase is also reported as a collision; this is used e.g. by CharacterBody3D for improving floor detection during floor snapping.

`max_collisions` allows to retrieve more than one collision result.

---

void **remove_collision_exception_with**(body: Node)

Removes a body from the list of bodies that this body can't collide with.

---

void **set_axis_lock**(axis: BodyAxis, lock: bool)

Locks or unlocks the specified linear or rotational `axis` depending on the value of `lock`.

---

bool **test_move**(from: Transform3D, motion: Vector3, collision: KinematicCollision3D = null, safe_margin: float = 0.001, recovery_as_collision: bool = false, max_collisions: int = 1)

Checks for collisions without moving the body. In order to be frame rate independent in Node._physics_process or Node._process, `motion` should be computed using `delta`.

Virtually sets the node's position, scale and rotation to that of the given Transform3D, then tries to move the body along the vector `motion`. Returns `true` if a collision would stop the body from moving along the whole path.

`collision` is an optional object of type KinematicCollision3D, which contains additional information about the collision when stopped, or when touching another body along the motion.

`safe_margin` is the extra margin used for collision recovery (see CharacterBody3D.safe_margin for more details).

If `recovery_as_collision` is `true`, any depenetration from the recovery phase is also reported as a collision; this is useful for checking whether the body would *touch* any other bodies.

`max_collisions` allows to retrieve more than one collision result.

PinJoint2D

Inherits: Joint2D **<** Node2D **<** CanvasItem **<** Node **<** Object

A physics joint that attaches two 2D physics bodies at a single point, allowing them to freely rotate.

Description

A physics joint that attaches two 2D physics bodies at a single point, allowing them to freely rotate. For example, a RigidBody2D can be attached to a StaticBody2D to create a pendulum or a seesaw.

Properties

bool	angular_limit_enabled	`false`
float	angular_limit_lower	`0.0`
float	angular_limit_upper	`0.0`
bool	motor_enabled	`false`
float	motor_target_velocity	`0.0`

float	softness	`0.0`

Property Descriptions

bool **angular_limit_enabled** = `false`

- void **set_angular_limit_enabled**(value: bool)
- bool **is_angular_limit_enabled**()

If `true`, the pin maximum and minimum rotation, defined by angular_limit_lower and angular_limit_upper are applied.

---

float **angular_limit_lower** = `0.0`

- void **set_angular_limit_lower**(value: float)
- float **get_angular_limit_lower**()

The minimum rotation. Only active if angular_limit_enabled is `true`.

---

float **angular_limit_upper** = `0.0`

- void **set_angular_limit_upper**(value: float)
- float **get_angular_limit_upper**()

The maximum rotation. Only active if angular_limit_enabled is `true`.

---

bool **motor_enabled** = `false`

- void **set_motor_enabled**(value: bool)
- bool **is_motor_enabled**()

When activated, a motor turns the pin.

---

float **motor_target_velocity** = `0.0`

- void **set_motor_target_velocity**(value: float)
- float **get_motor_target_velocity**()

Target speed for the motor. In radians per second.

---

float **softness** = `0.0`

- void **set_softness**(value: float)
- float **get_softness**()

The higher this value, the more the bond to the pinned partner can flex.

PinJoint3D

Inherits: Joint3D **<** Node3D **<** Node **<** Object

A physics joint that attaches two 3D physics bodies at a single point, allowing them to freely rotate.

Description

A physics joint that attaches two 3D physics bodies at a single point, allowing them to freely rotate. For example, a RigidBody3D can be attached to a StaticBody3D to create a pendulum or a seesaw.

Properties

float	params/bias	`0.3`
float	params/damping	`1.0`
float	params/impulse_clamp	`0.0`

Methods

float	get_param(param: Param) const

void set_param(param: Param, value: float)

Enumerations

enum **Param**: 🔗

Param **PARAM_BIAS** = `0`

The force with which the pinned objects stay in positional relation to each other. The higher, the stronger.

Param **PARAM_DAMPING** = `1`

The force with which the pinned objects stay in velocity relation to each other. The higher, the stronger.

Param **PARAM_IMPULSE_CLAMP** = `2`

If above 0, this value is the maximum value for an impulse that this Joint3D produces.

Property Descriptions

float **params/bias** = `0.3` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The force with which the pinned objects stay in positional relation to each other. The higher, the stronger.

float **params/damping** = `1.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The force with which the pinned objects stay in velocity relation to each other. The higher, the stronger.

---

float **params/impulse_clamp** = `0.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

If above 0, this value is the maximum value for an impulse that this Joint3D produces.

Method Descriptions

float **get_param**(param: Param) const

Returns the value of the specified parameter.

---

void **set_param**(param: Param, value: float)

Sets the value of the specified parameter.

PointLight2D

Inherits: Light2D **<** Node2D **<** CanvasItem **<** Node **<** Object

Positional 2D light source.

Description

Casts light in a 2D environment. This light's shape is defined by a (usually grayscale) texture.

Tutorials

- 2D lights and shadows

Properties

float	height	`0.0`
Vector2	offset	`Vector2(0, 0)`
Texture2D	texture	
float	texture_scale	`1.0`

Property Descriptions

float **height** = `0.0` 🔗

- void **set_height**(value: float)
- float **get_height**()

The height of the light. Used with 2D normal mapping. The units are in pixels, e.g. if the height is 100, then it will illuminate an object 100 pixels away at a 45° angle to the plane.

Vector2 **offset** = `Vector2(0, 0)` 🔗

- void **set_texture_offset**(value: Vector2)
- Vector2 **get_texture_offset**()

The offset of the light's texture.

Texture2D **texture** 🔗

- void **set_texture**(value: Texture2D)
- Texture2D **get_texture**()

Texture2D used for the light's appearance.

---

float **texture_scale** = `1.0` 🔗

- void **set_texture_scale**(value: float)
- float **get_texture_scale**()

The texture's scale factor.

Polygon2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A 2D polygon.

Description

A Polygon2D is defined by a set of points. Each point is connected to the next, with the final point being connected to the first, resulting in a closed polygon. Polygon2Ds can be filled with color (solid or gradient) or filled with a given texture.

Properties

bool	antialiased	`false`
Array	bones	`[]`
Color	color	`Color(1, 1, 1, 1)`
int	internal_vertex_count	`0`
float	invert_border	`100.0`
bool	invert_enabled	`false`

Vector2	offset	`Vector2(0, 0)`
PackedVector2Array	polygon	`PackedVector2Array()`
Array	polygons	`[]`
NodePath	skeleton	`NodePath("")`
Texture2D	texture	
Vector2	texture_offset	`Vector2(0, 0)`
float	texture_rotation	`0.0`
Vector2	texture_scale	`Vector2(1, 1)`
PackedVector2Array	uv	`PackedVector2Array()`
PackedColorArray	vertex_colors	`PackedColorArray()`

Methods

void	add_bone(path: NodePath, weights:

	PackedFloat32Array)
void	clear_bones()
void	erase_bone(index: int)
int	get_bone_count() const
NodePath	get_bone_path(index: int) const
PackedFloat32Array	get_bone_weights(index: int) const
void	set_bone_path(index: int, path: NodePath)
void	set_bone_weights(index: int, weights: PackedFloat32Array)

Property Descriptions

bool **antialiased** = `false` 🔗

- void **set_antialiased**(value: bool)
- bool **get_antialiased**()

If `true`, polygon edges will be anti-aliased.

Array **bones** = `[]` 🔗

Internal list of Bone2D nodes used by the assigned skeleton. Edited using the Polygon2D editor ("UV" button on the top toolbar).

Color **color** = `Color(1, 1, 1, 1)`

- void **set_color**(value: Color)
- Color **get_color**()

The polygon's fill color. If texture is set, it will be multiplied by this color. It will also be the default color for vertices not set in vertex_colors.

---

int **internal_vertex_count** = `0`

- void **set_internal_vertex_count**(value: int)
- int **get_internal_vertex_count**()

Number of internal vertices, used for UV mapping.

---

float **invert_border** = `100.0`

- void **set_invert_border**(value: float)
- float **get_invert_border**()

Added padding applied to the bounding box when invert_enabled is set to `true`. Setting this value too small may result in a "Bad Polygon" error.

---

bool **invert_enabled** = `false`

- void **set_invert_enabled**(value: bool)
- bool **get_invert_enabled**()

If `true`, the polygon will be inverted, containing the area outside the defined points and extending to the invert_border.

Vector2 **offset** = `Vector2(0, 0)` 🔗

- void **set_offset**(value: Vector2)
- Vector2 **get_offset**()

The offset applied to each vertex.

PackedVector2Array **polygon** = `PackedVector2Array()` 🔗

- void **set_polygon**(value: PackedVector2Array)
- PackedVector2Array **get_polygon**()

The polygon's list of vertices. The final point will be connected to the first.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

Array **polygons** = `[]` 🔗

- void **set_polygons**(value: Array)
- Array **get_polygons**()

The list of polygons, in case more than one is being represented. Every individual polygon is stored as a PackedInt32Array where each int is an index to a point in polygon. If empty, this property will be ignored, and the resulting single polygon will be composed of all points in polygon, using the order they are stored in.

NodePath **skeleton** = `NodePath("")` 🔗

- void **set_skeleton**(value: NodePath)
- NodePath **get_skeleton**()

Path to a Skeleton2D node used for skeleton-based deformations of this polygon. If empty or invalid, skeletal deformations will not be used.

Texture2D **texture** 🔗

- void **set_texture**(value: Texture2D)
- Texture2D **get_texture**()

The polygon's fill texture. Use uv to set texture coordinates.

Vector2 **texture_offset** = `Vector2(0, 0)` 🔗

- void **set_texture_offset**(value: Vector2)
- Vector2 **get_texture_offset**()

Amount to offset the polygon's texture. If set to `Vector2(0, 0)`, the texture's origin (its top-left corner) will be placed at the polygon's position.

---

float **texture_rotation** = `0.0` 🔗

- void **set_texture_rotation**(value: float)
- float **get_texture_rotation**()

The texture's rotation in radians.

Vector2 **texture_scale** = `Vector2(1, 1)` 🔗

- void **set_texture_scale**(value: Vector2)
- Vector2 **get_texture_scale**()

Amount to multiply the uv coordinates when using texture. Larger values make the texture smaller, and vice versa.

PackedVector2Array **uv** = `PackedVector2Array()` 🔗

- void **set_uv**(value: PackedVector2Array)
- PackedVector2Array **get_uv**()

Texture coordinates for each vertex of the polygon. There should be one UV value per polygon vertex. If there are fewer, undefined vertices will use `Vector2(0, 0)`.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

PackedColorArray **vertex_colors** = `PackedColorArray()` 🔗

- void **set_vertex_colors**(value: PackedColorArray)
- PackedColorArray **get_vertex_colors**()

Color for each vertex. Colors are interpolated between vertices, resulting in smooth gradients. There should be one per polygon vertex. If there are fewer, undefined vertices will use color.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedColorArray for more details.

Method Descriptions

void **add_bone**(path: NodePath, weights: PackedFloat32Array)

Adds a bone with the specified `path` and `weights`.

---

void **clear_bones**()

Removes all bones from this **Polygon2D**.

---

void **erase_bone**(index: int)

Removes the specified bone from this **Polygon2D**.

---

int **get_bone_count**() const

Returns the number of bones in this **Polygon2D**.

NodePath **get_bone_path**(index: int) const

Returns the path to the node associated with the specified bone.

PackedFloat32Array **get_bone_weights**(index: int) const

Returns the weight values of the specified bone.

---

void **set_bone_path**(index: int, path: NodePath)

Sets the path to the node associated with the specified bone.

---

void **set_bone_weights**(index: int, weights: PackedFloat32Array)

Sets the weight values for the specified bone.

Popup

Inherits: Window **<** Viewport **<** Node **<** Object

Inherited By: PopupMenu, PopupPanel

Base class for contextual windows and panels with fixed position.

Description

Popup is a base class for contextual windows and panels with fixed position. It's a modal by default (see Window.popup_window) and provides methods for implementing custom popup behavior.

Properties

bool	borderless	`true` (overrides Window)
bool	popup_window	`true` (overrides Window)
bool	transient	`true` (overrides Window)
bool	unresizable	`true` (overrides Window)
bool	visible	`false` (overrides Window)
bool	wrap_controls	`true` (overrides Window)

Signals

popup_hide()

Emitted when the popup is hidden.

PopupMenu

Inherits: Popup **<** Window **<** Viewport **<** Node **<** Object

A modal window used to display a list of options.

Description

PopupMenu is a modal window used to display a list of options. Useful for toolbars and context menus.

The size of a **PopupMenu** can be limited by using Window.max_size. If the height of the list of items is larger than the maximum height of the **PopupMenu**, a ScrollContainer within the popup will allow the user to scroll the contents. If no maximum size is set, or if it is set to `0`, the **PopupMenu** height will be limited by its parent rect.

All `set_*` methods allow negative item indices, i.e. `-1` to access the last item, `-2` to select the second-to-last item, and so on.

Incremental search: Like ItemList and Tree, **PopupMenu** supports searching within the list while the control is focused. Press a key that matches the first letter of an item's name to select the first item starting with the given letter. After that point, there are two ways to perform incremental search: 1) Press the same key again before the timeout duration to select the next item starting with the same letter. 2) Press letter keys that match the rest of the word before the timeout duration to match to select the item in question directly. Both of these actions will be reset to the beginning of the list if the timeout duration has passed since the last keystroke was registered. You can adjust the timeout duration by changing ProjectSettings.gui/timers/incremental_search_max_interval_msec.

Note: The ID values used for items are limited to 32 bits, not full 64 bits of int. This has a range of `-2^32` to `2^32 - 1`, i.e. `-2147483648` to `2147483647`.

Properties

bool	allow_search	`true`
bool	hide_on_checkable_item_selection	`true`
bool	hide_on_item_selection	`true`
bool	hide_on_state_item_selection	`false`
int	item_count	`0`
bool	prefer_native_menu	`false`
float	submenu_popup_delay	`0.3`
SystemMenus	system_menu_id	`0`

Methods

bool	activate_item_by_event(event: InputEvent, for_global_only: bool = false)
void	add_check_item(label: String, id: int = -1, accel: Key = 0)
void	add_check_shortcut(shortcut: Shortcut, id: int = -1, global: bool = false)
void	add_icon_check_item(texture: Texture2D, label: String, id: int = -1, accel: Key = 0)
void	add_icon_check_shortcut(texture: Texture2D, shortcut: Shortcut, id: int = -1, global: bool = false)
void	add_icon_item(texture: Texture2D, label: String, id: int = -1, accel: Key = 0)
void	add_icon_radio_check_item(texture: Texture2D, label: String, id: int = -1, accel: Key = 0)
void	add_icon_radio_check_shortcut(texture: Texture2D, shortcut: Shortcut, id: int = -1, global: bool = false)
void	add_icon_shortcut(texture: Texture2D, shortcut: Shortcut, id: int = -1, global: bool = false, allow_echo: bool = false)

void	add_item(label: String, id: int = -1, accel: Key = 0)
void	add_multistate_item(label: String, max_states: int, default_state: int = 0, id: int = -1, accel: Key = 0)
void	add_radio_check_item(label: String, id: int = -1, accel: Key = 0)
void	add_radio_check_shortcut(shortcut: Shortcut, id: int = -1, global: bool = false)
void	add_separator(label: String = "", id: int = -1)
void	add_shortcut(shortcut: Shortcut, id: int = -1, global: bool = false, allow_echo: bool = false)
void	add_submenu_item(label: String, submenu: String, id: int = -1)
void	add_submenu_node_item(label: String, submenu: PopupMenu, id: int = -1)
void	clear(free_submenus: bool = false)
int	get_focused_item() const
Key	get_item_accelerator(index: int) const

Texture2D	get_item_icon(index: int) const
int	get_item_icon_max_width(index: int) const
Color	get_item_icon_modulate(index: int) const
int	get_item_id(index: int) const
int	get_item_indent(index: int) const
int	get_item_index(id: int) const
String	get_item_language(index: int) const
Variant	get_item_metadata(index: int) const
int	get_item_multistate(index: int) const
int	get_item_multistate_max(index: int) const
Shortcut	get_item_shortcut(index: int) const
String	get_item_submenu(index: int) const

PopupMenu	get_item_submenu_node(index: int) const
String	get_item_text(index: int) const
TextDirection	get_item_text_direction(index: int) const
String	get_item_tooltip(index: int) const
bool	is_item_checkable(index: int) const
bool	is_item_checked(index: int) const
bool	is_item_disabled(index: int) const
bool	is_item_radio_checkable(index: int) const
bool	is_item_separator(index: int) const
bool	is_item_shortcut_disabled(index: int) const
bool	is_native_menu() const
bool	is_system_menu() const
void	remove_item(index: int)

void	scroll_to_item(index: int)
void	set_focused_item(index: int)
void	set_item_accelerator(index: int, accel: Key)
void	set_item_as_checkable(index: int, enable: bool)
void	set_item_as_radio_checkable(index: int, enable: bool)
void	set_item_as_separator(index: int, enable: bool)
void	set_item_checked(index: int, checked: bool)
void	set_item_disabled(index: int, disabled: bool)
void	set_item_icon(index: int, icon: Texture2D)
void	set_item_icon_max_width(index: int, width: int)
void	set_item_icon_modulate(index: int, modulate: Color)
void	set_item_id(index: int, id: int)

void	set_item_indent(index: int, indent: int)
void	set_item_language(index: int, language: String)
void	set_item_metadata(index: int, metadata: Variant)
void	set_item_multistate(index: int, state: int)
void	set_item_multistate_max(index: int, max_states: int)
void	set_item_shortcut(index: int, shortcut: Shortcut, global: bool = false)
void	set_item_shortcut_disabled(index: int, disabled: bool)
void	set_item_submenu(index: int, submenu: String)
void	set_item_submenu_node(index: int, submenu: PopupMenu)
void	set_item_text(index: int, text: String)
void	set_item_text_direction(index: int, direction: TextDirection)

void	set_item_tooltip(index: int, tooltip: String)
void	toggle_item_checked(index: int)
void	toggle_item_multistate(index: int)

Theme Properties

Color	font_accelerator_color	`Color(0.7, 0.7, 0.7, 0.8)`
Color	font_color	`Color(0.875, 0.875, 0.875, 1)`
Color	font_disabled_color	`Color(0.4, 0.4, 0.4, 0.8)`
Color	font_hover_color	`Color(0.875, 0.875, 0.875, 1)`
Color	font_outline_color	`Color(0, 0, 0, 1)`
Color	font_separator_color	`Color(0.875, 0.875, 0.875, 1)`

Color	font_separator_outline_color	`Color(0, 0, 0, 1)`
int	h_separation	`4`
int	icon_max_width	`0`
int	indent	`10`
int	item_end_padding	`2`
int	item_start_padding	`2`
int	outline_size	`0`
int	separator_outline_size	`0`
int	v_separation	`4`
Font	font	
Font	font_separator	
int	font_separator_size	
int	font_size	

Texture2D	checked
Texture2D	checked_disabled
Texture2D	radio_checked
Texture2D	radio_checked_disabled
Texture2D	radio_unchecked
Texture2D	radio_unchecked_disabled
Texture2D	submenu
Texture2D	submenu_mirrored
Texture2D	unchecked
Texture2D	unchecked_disabled
StyleBox	hover
StyleBox	labeled_separator_left

StyleBox	labeled_separator_right
StyleBox	panel
StyleBox	separator

Signals

id_focused(id: int) 🔗

Emitted when the user navigated to an item of some `id` using the ProjectSettings.input/ui_up or ProjectSettings.input/ui_down input action.

id_pressed(id: int) 🔗

Emitted when an item of some `id` is pressed or its accelerator is activated.

Note: If `id` is negative (either explicitly or due to overflow), this will return the corresponding index instead.

index_pressed(index: int) 🔗

Emitted when an item of some `index` is pressed or its accelerator is activated.

menu_changed() 🔗

Emitted when any item is added, modified or removed.

Property Descriptions

bool **allow_search** = `true` 🔗

- void **set_allow_search**(value: bool)
- bool **get_allow_search**()

If `true`, allows navigating **PopupMenu** with letter keys.

bool **hide_on_checkable_item_selection** = `true` 🔗

- void **set_hide_on_checkable_item_selection**(value: bool)
- bool **is_hide_on_checkable_item_selection**()

If `true`, hides the **PopupMenu** when a checkbox or radio button is selected.

bool **hide_on_item_selection** = `true` 🔗

- void **set_hide_on_item_selection**(value: bool)
- bool **is_hide_on_item_selection**()

If `true`, hides the **PopupMenu** when an item is selected.

bool **hide_on_state_item_selection** = `false` 🔗

- void **set_hide_on_state_item_selection**(value: bool)
- bool **is_hide_on_state_item_selection**()

If `true`, hides the **PopupMenu** when a state item is selected.

---

int **item_count** = `0` 🔗

- void **set_item_count**(value: int)
- int **get_item_count**()

The number of items currently in the list.

---

bool **prefer_native_menu** = `false` 🔗

- void **set_prefer_native_menu**(value: bool)
- bool **is_prefer_native_menu**()

If `true`, MenuBar will use native menu when supported.

Note: If **PopupMenu** is linked to StatusIndicator, MenuBar, or another **PopupMenu** item it can use native menu regardless of this property, use is_native_menu to check it.

---

float **submenu_popup_delay** = `0.3` 🔗

- void **set_submenu_popup_delay**(value: float)
- float **get_submenu_popup_delay**()

Sets the delay time in seconds for the submenu item to popup on mouse hovering. If the popup menu is added as a child of another (acting as a submenu), it will inherit the delay time of the parent menu item.

SystemMenus **system_menu_id** = `0` 🔗

- void **set_system_menu**(value: SystemMenus)
- SystemMenus **get_system_menu**()

If set to one of the values of SystemMenus, this **PopupMenu** is bound to the special system menu. Only one **PopupMenu** can be bound to each special menu at a time.

Method Descriptions

bool **activate_item_by_event**(event: InputEvent, for_global_only: bool = false)

Checks the provided `event` against the **PopupMenu**'s shortcuts and accelerators, and activates the first item with matching events. If `for_global_only` is `true`, only shortcuts and accelerators with `global` set to `true` will be called.

Returns `true` if an item was successfully activated.

Note: Certain Controls, such as MenuButton, will call this method automatically.

---

void **add_check_item**(label: String, id: int = -1, accel: Key = 0)

Adds a new checkable item with text `label`.

An `id` can optionally be provided, as well as an accelerator (`accel`). If no `id` is provided, one will be created from the index. If no `accel` is provided, then the default value of 0 (corresponding to @GlobalScope.KEY_NONE) will be assigned to the item (which means it won't have any accelerator). See get_item_accelerator for more info on accelerators.

Note: Checkable items just display a checkmark, but don't have any built-in checking behavior and must be checked/unchecked manually. See set_item_checked for more info on how to control it.

---

void **add_check_shortcut**(shortcut: Shortcut, id: int = -1, global: bool = false) 🔗

Adds a new checkable item and assigns the specified Shortcut to it. Sets the label of the checkbox to the Shortcut's name.

An `id` can optionally be provided. If no `id` is provided, one will be created from the index.

Note: Checkable items just display a checkmark, but don't have any built-in checking behavior and must be checked/unchecked manually. See set_item_checked for more info on how to control it.

---

void **add_icon_check_item**(texture: Texture2D, label: String, id: int = -1, accel: Key = 0) 🔗

Adds a new checkable item with text `label` and icon `texture`.

An `id` can optionally be provided, as well as an accelerator (`accel`). If no `id` is provided, one will be created from the index. If no `accel` is provided, then the default value of 0 (corresponding to @GlobalScope.KEY_NONE) will be assigned to the item (which means it won't have any accelerator). See get_item_accelerator for more info on accelerators.

Note: Checkable items just display a checkmark, but don't have any built-in checking behavior and must be checked/unchecked manually. See set_item_checked for more info on how to control it.

---

void **add_icon_check_shortcut**(texture: Texture2D, shortcut: Shortcut, id: int = -1, global: bool = false) 🔗

Adds a new checkable item and assigns the specified Shortcut and icon `texture` to it. Sets the label of the checkbox to the Shortcut's name.

An `id` can optionally be provided. If no `id` is provided, one will be created from the index.

Note: Checkable items just display a checkmark, but don't have any built-in checking behavior and must be checked/unchecked manually. See set_item_checked for more info on how to control it.

---

void **add_icon_item**(texture: Texture2D, label: String, id: int = -1, accel: Key = 0)

Adds a new item with text `label` and icon `texture`.

An `id` can optionally be provided, as well as an accelerator (`accel`). If no `id` is provided, one will be created from the index. If no `accel` is provided, then the default value of 0 (corresponding to @GlobalScope.KEY_NONE) will be assigned to the item (which means it won't have any accelerator). See get_item_accelerator for more info on accelerators.

---

void **add_icon_radio_check_item**(texture: Texture2D, label: String, id: int = -1, accel: Key = 0)

Same as add_icon_check_item, but uses a radio check button.

---

void **add_icon_radio_check_shortcut**(texture: Texture2D, shortcut: Shortcut, id: int = -1, global: bool = false)

Same as add_icon_check_shortcut, but uses a radio check button.

---

void **add_icon_shortcut**(texture: Texture2D, shortcut: Shortcut, id: int = -1, global: bool = false, allow_echo: bool = false) 🔗

Adds a new item and assigns the specified Shortcut and icon `texture` to it. Sets the label of the checkbox to the Shortcut's name.

An `id` can optionally be provided. If no `id` is provided, one will be created from the index.

If `allow_echo` is `true`, the shortcut can be activated with echo events.

---

void **add_item**(label: String, id: int = -1, accel: Key = 0) 🔗

Adds a new item with text `label`.

An `id` can optionally be provided, as well as an accelerator (`accel`). If no `id` is provided, one will be created from the index. If no `accel` is provided, then the default value of 0 (corresponding to @GlobalScope.KEY_NONE) will be assigned to the item (which means it won't have any accelerator). See get_item_accelerator for more info on accelerators.

Note: The provided `id` is used only in id_pressed and id_focused signals. It's not related to the `index` arguments in e.g. set_item_checked.

---

void **add_multistate_item**(label: String, max_states: int, default_state: int = 0, id: int = -1, accel: Key = 0) 🔗

Adds a new multistate item with text `label`.

Contrarily to normal binary items, multistate items can have more than two states, as defined by `max_states`. The default value is defined by `default_state`.

An `id` can optionally be provided, as well as an accelerator (`accel`). If no `id` is provided, one will be created from the index. If no `accel` is provided, then the default value of 0 (corresponding to @GlobalScope.KEY_NONE) will be assigned to the item (which means it won't have any accelerator). See get_item_accelerator for more info on accelerators.

Note: Multistate items don't update their state automatically and must be done manually. See toggle_item_multistate, set_item_multistate and get_item_multistate for more info on how to control it.

Example usage:

```
func _ready():
    add_multistate_item("Item", 3, 0)

    index_pressed.connect(func(index: int):
            toggle_item_multistate(index)
            match get_item_multistate(index):
                0:
                    print("First state")
                1:
                    print("Second state")
                2:
                    print("Third state")
        )
```

---

void **add_radio_check_item**(label: String, id: int = -1, accel: Key = 0)

Adds a new radio check button with text `label`.

An `id` can optionally be provided, as well as an accelerator (`accel`). If no `id` is provided, one will be created from the index. If no `accel` is provided, then the default value of 0 (corresponding to

@GlobalScope.KEY_NONE) will be assigned to the item (which means it won't have any accelerator). See get_item_accelerator for more info on accelerators.

Note: Checkable items just display a checkmark, but don't have any built-in checking behavior and must be checked/unchecked manually. See set_item_checked for more info on how to control it.

---

void **add_radio_check_shortcut**(shortcut: Shortcut, id: int = -1, global: bool = false) 🔗

Adds a new radio check button and assigns a Shortcut to it. Sets the label of the checkbox to the Shortcut's name.

An `id` can optionally be provided. If no `id` is provided, one will be created from the index.

Note: Checkable items just display a checkmark, but don't have any built-in checking behavior and must be checked/unchecked manually. See set_item_checked for more info on how to control it.

---

void **add_separator**(label: String = "", id: int = -1) 🔗

Adds a separator between items. Separators also occupy an index, which you can set by using the `id` parameter.

A `label` can optionally be provided, which will appear at the center of the separator.

---

void **add_shortcut**(shortcut: Shortcut, id: int = -1, global: bool = false, allow_echo: bool = false) 🔗

Adds a Shortcut.

An `id` can optionally be provided. If no `id` is provided, one will be created from the index.

If `allow_echo` is `true`, the shortcut can be activated with echo events.

---

void **add_submenu_item**(label: String, submenu: String, id: int = -1) 🔗

Deprecated: Prefer using add_submenu_node_item instead.

Adds an item that will act as a submenu of the parent **PopupMenu** node when clicked. The `submenu` argument must be the name of an existing **PopupMenu** that has been added as a child to this node. This submenu will be shown when the item is clicked, hovered for long enough, or activated using the `ui_select` or `ui_right` input actions.

An `id` can optionally be provided. If no `id` is provided, one will be created from the index.

---

void **add_submenu_node_item**(label: String, submenu: PopupMenu, id: int = -1) 🔗

Adds an item that will act as a submenu of the parent **PopupMenu** node when clicked. This submenu will be shown when the item is clicked, hovered for long enough, or activated using the `ui_select` or `ui_right` input actions.

`submenu` must be either child of this **PopupMenu** or has no parent node (in which case it will be automatically added as a child). If the `submenu` popup has another parent, this method will fail.

An `id` can optionally be provided. If no `id` is provided, one will be created from the index.

---

void **clear**(free_submenus: bool = false) 🔗

Removes all items from the **PopupMenu**. If `free_submenus` is `true`, the submenu nodes are automatically freed.

---

int **get_focused_item**() const 🔗

Returns the index of the currently focused item. Returns `-1` if no item is focused.

Key **get_item_accelerator**(index: int) const 🔗

Returns the accelerator of the item at the given `index`. An accelerator is a keyboard shortcut that can be pressed to trigger the menu button even if it's not currently open. The return value is an integer which is generally a combination of KeyModifierMasks and Keys using bitwise OR such as `KEY_MASK_CTRL | KEY_A` (`Ctrl + A`). If no accelerator is defined for the specified `index`, get_item_accelerator returns `0` (corresponding to @GlobalScope.KEY_NONE).

Texture2D **get_item_icon**(index: int) const 🔗

Returns the icon of the item at the given `index`.

---

int **get_item_icon_max_width**(index: int) const 🔗

Returns the maximum allowed width of the icon for the item at the given `index`.

Color **get_item_icon_modulate**(index: int) const

Returns a Color modulating the item's icon at the given `index`.

---

int **get_item_id**(index: int) const

Returns the ID of the item at the given `index`. `id` can be manually assigned, while index can not.

---

int **get_item_indent**(index: int) const

Returns the horizontal offset of the item at the given `index`.

---

int **get_item_index**(id: int) const

Returns the index of the item containing the specified `id`. Index is automatically assigned to each item by the engine and can not be set manually.

String **get_item_language**(index: int) const

Returns item's text language code.

Variant **get_item_metadata**(index: int) const

Returns the metadata of the specified item, which might be of any type. You can set it with set_item_metadata, which provides a simple way of assigning context data to items.

---

int **get_item_multistate**(index: int) const

Returns the state of the item at the given `index`.

---

int **get_item_multistate_max**(index: int) const

Returns the max states of the item at the given `index`.

Shortcut **get_item_shortcut**(index: int) const

Returns the Shortcut associated with the item at the given `index`.

String **get_item_submenu**(index: int) const

Deprecated: Prefer using get_item_submenu_node instead.

Returns the submenu name of the item at the given `index`. See add_submenu_item for more info on how to add a submenu.

PopupMenu **get_item_submenu_node**(index: int) const

Returns the submenu of the item at the given `index`, or `null` if no submenu was added. See add_submenu_node_item for more info on how to add a submenu.

String **get_item_text**(index: int) const

Returns the text of the item at the given `index`.

TextDirection **get_item_text_direction**(index: int) const

Returns item's text base writing direction.

String **get_item_tooltip**(index: int) const

Returns the tooltip associated with the item at the given `index`.

bool **is_item_checkable**(index: int) const

Returns `true` if the item at the given `index` is checkable in some way, i.e. if it has a checkbox or radio button.

Note: Checkable items just display a checkmark or radio button, but don't have any built-in checking behavior and must be checked/unchecked manually.

bool **is_item_checked**(index: int) const

Returns `true` if the item at the given `index` is checked.

bool **is_item_disabled**(index: int) const

Returns `true` if the item at the given `index` is disabled. When it is disabled it can't be selected, or its action invoked.

See set_item_disabled for more info on how to disable an item.

bool **is_item_radio_checkable**(index: int) const

Returns `true` if the item at the given `index` has radio button-style checkability.

Note: This is purely cosmetic; you must add the logic for checking/unchecking items in radio groups.

---

bool **is_item_separator**(index: int) const

Returns `true` if the item is a separator. If it is, it will be displayed as a line. See add_separator for more info on how to add a separator.

---

bool **is_item_shortcut_disabled**(index: int) const

Returns `true` if the specified item's shortcut is disabled.

---

bool **is_native_menu**() const

Returns `true` if the system native menu is supported and currently used by this **PopupMenu**.

---

bool **is_system_menu**() const

Returns `true` if the menu is bound to the special system menu.

---

void **remove_item**(index: int)

Removes the item at the given `index` from the menu.

Note: The indices of items after the removed item will be shifted by one.

---

void **scroll_to_item**(index: int)

Moves the scroll view to make the item at the given `index` visible.

---

void **set_focused_item**(index: int) 🔗

Sets the currently focused item as the given `index`.

Passing `-1` as the index makes so that no item is focused.

---

void **set_item_accelerator**(index: int, accel: Key) 🔗

Sets the accelerator of the item at the given `index`. An accelerator is a keyboard shortcut that can be pressed to trigger the menu button even if it's not currently open. `accel` is generally a combination of KeyModifierMasks and Keys using bitwise OR such as `KEY_MASK_CTRL | KEY_A` (`Ctrl + A`).

---

void **set_item_as_checkable**(index: int, enable: bool) 🔗

Sets whether the item at the given `index` has a checkbox. If `false`, sets the type of the item to plain text.

Note: Checkable items just display a checkmark, but don't have any built-in checking behavior and must be checked/unchecked manually.

---

void **set_item_as_radio_checkable**(index: int, enable: bool) 🔗

Sets the type of the item at the given `index` to radio button. If `false`, sets the type of the item to plain text.

---

void **set_item_as_separator**(index: int, enable: bool) 🔗

Mark the item at the given `index` as a separator, which means that it would be displayed as a line. If `false`, sets the type of the item to plain text.

void **set_item_checked**(index: int, checked: bool) 🔗

Sets the checkstate status of the item at the given `index`.

void **set_item_disabled**(index: int, disabled: bool) 🔗

Enables/disables the item at the given `index`. When it is disabled, it can't be selected and its action can't be invoked.

void **set_item_icon**(index: int, icon: Texture2D) 🔗

Replaces the Texture2D icon of the item at the given `index`.

void **set_item_icon_max_width**(index: int, width: int) 🔗

Sets the maximum allowed width of the icon for the item at the given `index`. This limit is applied on top of the default size of the icon and on top of icon_max_width. The height is adjusted according to the icon's ratio.

void **set_item_icon_modulate**(index: int, modulate: Color) 🔗

Sets a modulating Color of the item's icon at the given `index`.

void **set_item_id**(index: int, id: int) 🔗

Sets the `id` of the item at the given `index`.

The `id` is used in id_pressed and id_focused signals.

---

void **set_item_indent**(index: int, indent: int)

Sets the horizontal offset of the item at the given `index`.

---

void **set_item_language**(index: int, language: String)

Sets language code of item's text used for line-breaking and text shaping algorithms, if left empty current locale is used instead.

---

void **set_item_metadata**(index: int, metadata: Variant)

Sets the metadata of an item, which may be of any type. You can later get it with get_item_metadata, which provides a simple way of assigning context data to items.

---

void **set_item_multistate**(index: int, state: int)

Sets the state of a multistate item. See add_multistate_item for details.

---

void **set_item_multistate_max**(index: int, max_states: int)

Sets the max states of a multistate item. See add_multistate_item for details.

---

void **set_item_shortcut**(index: int, shortcut: Shortcut, global: bool = false)

Sets a Shortcut for the item at the given `index`.

void **set_item_shortcut_disabled**(index: int, disabled: bool)

Disables the Shortcut of the item at the given `index`.

void **set_item_submenu**(index: int, submenu: String)

Deprecated: Prefer using set_item_submenu_node instead.

Sets the submenu of the item at the given `index`. The submenu is the name of a child **PopupMenu** node that would be shown when the item is clicked.

void **set_item_submenu_node**(index: int, submenu: PopupMenu)

Sets the submenu of the item at the given `index`. The submenu is a **PopupMenu** node that would be shown when the item is clicked. It must either be a child of this **PopupMenu** or has no parent (in which case it will be automatically added as a child). If the `submenu` popup has another parent, this method will fail.

void **set_item_text**(index: int, text: String)

Sets the text of the item at the given `index`.

void **set_item_text_direction**(index: int, direction: TextDirection)

Sets item's text base writing direction.

void **set_item_tooltip**(index: int, tooltip: String)

Sets the String tooltip of the item at the given `index`.

void **toggle_item_checked**(index: int)

Toggles the check state of the item at the given `index`.

void **toggle_item_multistate**(index: int)

Cycle to the next state of a multistate item. See add_multistate_item for details.

Theme Property Descriptions

Color **font_accelerator_color** = `Color(0.7, 0.7, 0.7, 0.8)`

The text Color used for shortcuts and accelerators that show next to the menu item name when defined. See get_item_accelerator for more info on accelerators.

Color **font_color** = `Color(0.875, 0.875, 0.875, 1)`

The default text Color for menu items' names.

Color **font_disabled_color** = `Color(0.4, 0.4, 0.4, 0.8)`

Color used for disabled menu items' text.

Color **font_hover_color** = `Color(0.875, 0.875, 0.875, 1)`

Color used for the hovered text.

Color **font_outline_color** = `Color(0, 0, 0, 1)`

The tint of text outline of the menu item.

Color **font_separator_color** = `Color(0.875, 0.875, 0.875, 1)`

Color used for labeled separators' text. See add_separator.

Color **font_separator_outline_color** = `Color(0, 0, 0, 1)`

The tint of text outline of the labeled separator.

---

int **h_separation** = `4`

The horizontal space between the item's elements.

---

int **icon_max_width** = `0`

The maximum allowed width of the item's icon. This limit is applied on top of the default size of the icon, but before the value set with set_item_icon_max_width. The height is adjusted according to the icon's ratio.

---

int **indent** = `10`

Width of the single indentation level.

---

int **item_end_padding** = `2`

Horizontal padding to the right of the items (or left, in RTL layout).

---

int **item_start_padding** = `2`

Horizontal padding to the left of the items (or right, in RTL layout).

---

int **outline_size** = `0`

The size of the item text outline.

Note: If using a font with FontFile.multichannel_signed_distance_field enabled, its FontFile.msdf_pixel_range must be set to at least *twice* the value of outline_size for outline rendering to look correct. Otherwise, the outline may appear to be cut off earlier than intended.

---

int **separator_outline_size** = `0`

The size of the labeled separator text outline.

---

int **v_separation** = `4`

The vertical space between each menu item.

Font **font**

Font used for the menu items.

Font **font_separator**

Font used for the labeled separator.

int **font_separator_size** 🔗

Font size of the labeled separator.

int **font_size** 🔗

Font size of the menu items.

Texture2D **checked** 🔗

Texture2D icon for the checked checkbox items.

Texture2D **checked_disabled** 🔗

Texture2D icon for the checked checkbox items when they are disabled.

Texture2D **radio_checked** 🔗

Texture2D icon for the checked radio button items.

Texture2D **radio_checked_disabled** 🔗

Texture2D icon for the checked radio button items when they are disabled.

Texture2D **radio_unchecked** 🔗

Texture2D icon for the unchecked radio button items.

Texture2D **radio_unchecked_disabled** 🔗

Texture2D icon for the unchecked radio button items when they are disabled.

Texture2D **submenu** 🔗

Texture2D icon for the submenu arrow (for left-to-right layouts).

Texture2D **submenu_mirrored** 🔗

Texture2D icon for the submenu arrow (for right-to-left layouts).

Texture2D **unchecked** 🔗

Texture2D icon for the unchecked checkbox items.

Texture2D **unchecked_disabled** 🔗

Texture2D icon for the unchecked checkbox items when they are disabled.

StyleBox **hover** 🔗

StyleBox displayed when the **PopupMenu** item is hovered.

StyleBox **labeled_separator_left** 🔗

StyleBox for the left side of labeled separator. See add_separator.

StyleBox labeled_separator_right

StyleBox for the right side of labeled separator. See add_separator.

StyleBox panel

StyleBox for the the background panel.

StyleBox separator

StyleBox used for the separators. See add_separator.

PopupPanel

Inherits: Popup **<** Window **<** Viewport **<** Node **<** Object

A popup with a panel background.

Description

A popup with a configurable panel background. Any child controls added to this node will be stretched to fit the panel's size (similar to how PanelContainer works). If you are making windows, see Window.

Theme Properties

StyleBox	panel

Theme Property Descriptions

StyleBox **panel**

StyleBox for the the background panel.

ProgressBar

Inherits: Range **<** Control **<** CanvasItem **<** Node **<** Object

A control used for visual representation of a percentage.

Description

A control used for visual representation of a percentage. Shows fill percentage from right to left.

Properties

bool	editor_preview_indeterminate	
int	fill_mode	`0`
bool	indeterminate	`false`
bool	show_percentage	`true`

Theme Properties

Color	font_color	`Color(0.95, 0.95, 0.95, 1)`

Color	font_outline_color	`Color(0, 0, 0, 1)`
int	outline_size	`0`
Font	font	
int	font_size	
StyleBox	background	
StyleBox	fill	

Enumerations

enum **FillMode**: 🔗

FillMode **FILL_BEGIN_TO_END** = `0`

The progress bar fills from begin to end horizontally, according to the language direction. If Control.is_layout_rtl returns `false`, it fills from left to right, and if it returns `true`, it fills from right to left.

FillMode **FILL_END_TO_BEGIN** = `1`

The progress bar fills from end to begin horizontally, according to the language direction. If Control.is_layout_rtl returns `false`, it fills from right to left, and if it returns `true`, it fills from left to right.

FillMode **FILL_TOP_TO_BOTTOM** = `2`

The progress fills from top to bottom.

FillMode **FILL_BOTTOM_TO_TOP** = `3`

The progress fills from bottom to top.

Property Descriptions

bool **editor_preview_indeterminate** 🔗

- void **set_editor_preview_indeterminate**(value: bool)
- bool **is_editor_preview_indeterminate_enabled**()

If `false`, the indeterminate animation will be paused in the editor.

---

int **fill_mode** = `0` 🔗

- void **set_fill_mode**(value: int)
- int **get_fill_mode**()

The fill direction. See FillMode for possible values.

---

bool **indeterminate** = `false` 🔗

- void **set_indeterminate**(value: bool)
- bool **is_indeterminate**()

When set to `true`, the progress bar indicates that something is happening with an animation, but does not show the fill percentage or value.

---

bool **show_percentage** = `true` 🔗

- void **set_show_percentage**(value: bool)
- bool **is_percentage_shown**()

If `true`, the fill percentage is displayed on the bar.

Theme Property Descriptions

Color **font_color** = `Color(0.95, 0.95, 0.95, 1)` 🔗

The color of the text.

Color **font_outline_color** = `Color(0, 0, 0, 1)` 🔗

The tint of text outline of the **ProgressBar**.

int **outline_size** = `0` 🔗

The size of the text outline.

Note: If using a font with FontFile.multichannel_signed_distance_field enabled, its FontFile.msdf_pixel_range must be set to at least *twice* the value of outline_size for outline rendering to look correct. Otherwise, the outline may appear to be cut off earlier than intended.

Font **font** 🔗

Font used to draw the fill percentage if show_percentage is `true`.

int **font_size** 🔗

Font size used to draw the fill percentage if show_percentage is `true`.

StyleBox **background** 🔗

The style of the background.

StyleBox **fill** 🔗

The style of the progress (i.e. the part that fills the bar).

Range

Inherits: Control **<** CanvasItem **<** Node **<** Object

Inherited By: EditorSpinSlider, ProgressBar, ScrollBar, Slider, SpinBox, TextureProgressBar

Abstract base class for controls that represent a number within a range.

Description

Range is an abstract base class for controls that represent a number within a range, using a configured step and page size. See e.g. ScrollBar and Slider for examples of higher-level nodes using Range.

Properties

bool	allow_greater	`false`
bool	allow_lesser	`false`
bool	exp_edit	`false`
float	max_value	`100.0`
float	min_value	`0.0`

float	page	`0.0`
float	ratio	
bool	rounded	`false`
BitField[SizeFlags]	size_flags_vertical	`0` (overrides Control)
float	step	`0.01`
float	value	`0.0`

Methods

void	_value_changed(new_value: float) virtual
void	set_value_no_signal(value: float)
void	share(with: Node)
void	unshare()

Signals

changed()

Emitted when min_value, max_value, page, or step change.

---

value_changed(value: float)

Emitted when value changes. When used on a Slider, this is called continuously while dragging (potentially every frame). If you are performing an expensive operation in a function connected to value_changed, consider using a *debouncing* Timer to call the function less often.

Note: Unlike signals such as LineEdit.text_changed, value_changed is also emitted when `value` is set directly via code.

Property Descriptions

bool **allow_greater** = `false`

- void **set_allow_greater**(value: bool)
- bool **is_greater_allowed**()

If `true`, value may be greater than max_value.

---

bool **allow_lesser** = `false`

- void **set_allow_lesser**(value: bool)
- bool **is_lesser_allowed**()

If `true`, value may be less than min_value.

---

bool **exp_edit** = `false`

- void **set_exp_ratio**(value: bool)
- bool **is_ratio_exp**()

If `true`, and min_value is greater than 0, value will be represented exponentially rather than linearly.

---

float **max_value** = `100.0`

- void **set_max**(value: float)
- float **get_max**()

Maximum value. Range is clamped if value is greater than max_value.

---

float **min_value** = `0.0`

- void **set_min**(value: float)
- float **get_min**()

Minimum value. Range is clamped if value is less than min_value.

---

float **page** = `0.0`

- void **set_page**(value: float)
- float **get_page**()

Page size. Used mainly for ScrollBar. ScrollBar's length is its size multiplied by page over the difference between min_value and max_value.

---

float **ratio**

- void **set_as_ratio**(value: float)

- float **get_as_ratio**()

The value mapped between 0 and 1.

---

float **rounded** = `false` 🔗

- void **set_use_rounded_values**(value: bool)
- bool **is_using_rounded_values**()

If `true`, value will always be rounded to the nearest integer.

---

float **step** = `0.01` 🔗

- void **set_step**(value: float)
- float **get_step**()

If greater than 0, value will always be rounded to a multiple of this property's value. If rounded is also `true`, value will first be rounded to a multiple of this property's value, then rounded to the nearest integer.

---

float **value** = `0.0` 🔗

- void **set_value**(value: float)
- float **get_value**()

Range's current value. Changing this property (even via code) will trigger value_changed signal. Use set_value_no_signal if you want to avoid it.

Method Descriptions

void **_value_changed**(new_value: float) virtual

Called when the **Range**'s value is changed (following the same conditions as value_changed).

---

void **set_value_no_signal**(value: float)

Sets the **Range**'s current value to the specified `value`, without emitting the value_changed signal.

---

void **share**(with: Node)

Binds two **Range**s together along with any ranges previously grouped with either of them. When any of range's member variables change, it will share the new value with all other ranges in its group.

---

void **unshare**()

Stops the **Range** from sharing its member variables with any other.

RayCast2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A ray in 2D space, used to find the first CollisionObject2D it intersects.

Description

A raycast represents a ray from its origin to its target_position that finds the closest CollisionObject2D along its path, if it intersects any.

RayCast2D can ignore some objects by adding them to an exception list, by making its detection reporting ignore Area2Ds (collide_with_areas) or PhysicsBody2Ds (collide_with_bodies), or by configuring physics layers.

RayCast2D calculates intersection every physics frame, and it holds the result until the next physics frame. For an immediate raycast, or if you want to configure a **RayCast2D** multiple times within the same physics frame, use force_raycast_update.

To sweep over a region of 2D space, you can approximate the region with multiple **RayCast2D**s or use ShapeCast2D.

Tutorials

- Ray-casting

Properties

bool	collide_with_areas	`false`
bool	collide_with_bodies	`true`
int	collision_mask	`1`
bool	enabled	`true`
bool	exclude_parent	`true`
bool	hit_from_inside	`false`
Vector2	target_position	`Vector2(0, 50)`

Methods

void	add_exception(node: CollisionObject2D)
void	add_exception_rid(rid: RID)
void	clear_exceptions()
void	force_raycast_update()

Object	get_collider() const
RID	get_collider_rid() const
int	get_collider_shape() const
bool	get_collision_mask_value(layer_number: int) const
Vector2	get_collision_normal() const
Vector2	get_collision_point() const
bool	is_colliding() const
void	remove_exception(node: CollisionObject2D)
void	remove_exception_rid(rid: RID)
void	set_collision_mask_value(layer_number: int, value: bool)

Property Descriptions

bool **collide_with_areas** = `false` 🔗

- void **set_collide_with_areas**(value: bool)

- bool **is_collide_with_areas_enabled**()

If `true`, collisions with Area2Ds will be reported.

bool **collide_with_bodies** = `true`

- void **set_collide_with_bodies**(value: bool)
- bool **is_collide_with_bodies_enabled**()

If `true`, collisions with PhysicsBody2Ds will be reported.

int **collision_mask** = `1`

- void **set_collision_mask**(value: int)
- int **get_collision_mask**()

The ray's collision mask. Only objects in at least one collision layer enabled in the mask will be detected. See Collision layers and masks in the documentation for more information.

bool **enabled** = `true`

- void **set_enabled**(value: bool)
- bool **is_enabled**()

If `true`, collisions will be reported.

bool **exclude_parent** = `true`

- void **set_exclude_parent_body**(value: bool)
- bool **get_exclude_parent_body**()

If `true`, the parent node will be excluded from collision detection.

---

bool **hit_from_inside** = `false`

- void **set_hit_from_inside**(value: bool)
- bool **is_hit_from_inside_enabled**()

If `true`, the ray will detect a hit when starting inside shapes. In this case the collision normal will be `Vector2(0, 0)`. Does not affect concave polygon shapes.

Vector2 **target_position** = `Vector2(0, 50)`

- void **set_target_position**(value: Vector2)
- Vector2 **get_target_position**()

The ray's destination point, relative to the RayCast's `position`.

Method Descriptions

void **add_exception**(node: CollisionObject2D)

Adds a collision exception so the ray does not report collisions with the specified CollisionObject2D node.

---

void **add_exception_rid**(rid: RID)

Adds a collision exception so the ray does not report collisions with the specified RID.

---

void **clear_exceptions**()

Removes all collision exceptions for this ray.

---

void **force_raycast_update**() 🔗

Updates the collision information for the ray immediately, without waiting for the next `_physics_process` call. Use this method, for example, when the ray or its parent has changed state.

Note: enabled does not need to be `true` for this to work.

Object **get_collider**() const 🔗

Returns the first object that the ray intersects, or `null` if no object is intersecting the ray (i.e. is_colliding returns `false`).

RID **get_collider_rid**() const 🔗

Returns the RID of the first object that the ray intersects, or an empty RID if no object is intersecting the ray (i.e. is_colliding returns `false`).

---

int **get_collider_shape**() const 🔗

Returns the shape ID of the first object that the ray intersects, or `0` if no object is intersecting the ray (i.e. is_colliding returns `false`).

To get the intersected shape node, for a CollisionObject2D target, use:

GDScript

```
var target = get_collider() # A CollisionObject2D.
var shape_id = get_collider_shape() # The shape index in the
collider.
var owner_id = target.shape_find_owner(shape_id) # The owner ID
```

```
in the collider.
var shape = target.shape_owner_get_owner(owner_id)
```

C#

```
var target = (CollisionObject2D)GetCollider(); // A
CollisionObject2D.
var shapeId = GetColliderShape(); // The shape index in the
collider.
var ownerId = target.ShapeFindOwner(shapeId); // The owner ID
in the collider.
var shape = target.ShapeOwnerGetOwner(ownerId);
```

---

bool **get_collision_mask_value**(layer_number: int) const 🔗

Returns whether or not the specified layer of the collision_mask is enabled, given a `layer_number` between 1 and 32.

Vector2 **get_collision_normal**() const 🔗

Returns the normal of the intersecting object's shape at the collision point, or `Vector2(0, 0)` if the ray starts inside the shape and hit_from_inside is `true`.

Note: Check that is_colliding returns `true` before calling this method to ensure the returned normal is valid and up-to-date.

Vector2 **get_collision_point**() const 🔗

Returns the collision point at which the ray intersects the closest object, in the global coordinate system. If hit_from_inside is `true` and the ray starts inside of a collision shape, this function will return the origin point of the ray.

Note: Check that is_colliding returns `true` before calling this method to ensure the returned point is valid and up-to-date.

bool **is_colliding**() const

Returns whether any object is intersecting with the ray's vector (considering the vector length).

void **remove_exception**(node: CollisionObject2D)

Removes a collision exception so the ray does report collisions with the specified CollisionObject2D node.

void **remove_exception_rid**(rid: RID)

Removes a collision exception so the ray does report collisions with the specified RID.

void **set_collision_mask_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the collision_mask, given a `layer_number` between 1 and 32.

RayCast3D

Inherits: Node3D **<** Node **<** Object

A ray in 3D space, used to find the first CollisionObject3D it intersects.

Description

A raycast represents a ray from its origin to its target_position that finds the closest CollisionObject3D along its path, if it intersects any.

RayCast3D can ignore some objects by adding them to an exception list, by making its detection reporting ignore Area3Ds (collide_with_areas) or PhysicsBody3Ds (collide_with_bodies), or by configuring physics layers.

RayCast3D calculates intersection every physics frame, and it holds the result until the next physics frame. For an immediate raycast, or if you want to configure a **RayCast3D** multiple times within the same physics frame, use force_raycast_update.

To sweep over a region of 3D space, you can approximate the region with multiple **RayCast3D**s or use ShapeCast3D.

Tutorials

- Ray-casting
- 3D Voxel Demo [https://godotengine.org/asset-library/asset/2755]

Properties

bool	collide_with_areas	`false`
bool	collide_with_bodies	`true`
int	collision_mask	`1`
Color	debug_shape_custom_color	`Color(0, 0, 0, 1)`
int	debug_shape_thickness	`2`
bool	enabled	`true`
bool	exclude_parent	`true`
bool	hit_back_faces	`true`
bool	hit_from_inside	`false`
Vector3	target_position	`Vector3(0, -1, 0)`

Methods

void	add_exception(node: CollisionObject3D)

void	add_exception_rid(rid: RID)
void	clear_exceptions()
void	force_raycast_update()
Object	get_collider() const
RID	get_collider_rid() const
int	get_collider_shape() const
int	get_collision_face_index() const
bool	get_collision_mask_value(layer_number: int) const
Vector3	get_collision_normal() const
Vector3	get_collision_point() const
bool	is_colliding() const
void	remove_exception(node: CollisionObject3D)
void	remove_exception_rid(rid: RID)

void	set_collision_mask_value(layer_number: int, value: bool)

Property Descriptions

bool **collide_with_areas** = `false`

- void **set_collide_with_areas**(value: bool)
- bool **is_collide_with_areas_enabled**()

If `true`, collisions with Area3Ds will be reported.

---

bool **collide_with_bodies** = `true`

- void **set_collide_with_bodies**(value: bool)
- bool **is_collide_with_bodies_enabled**()

If `true`, collisions with PhysicsBody3Ds will be reported.

---

int **collision_mask** = `1`

- void **set_collision_mask**(value: int)
- int **get_collision_mask**()

The ray's collision mask. Only objects in at least one collision layer enabled in the mask will be detected. See Collision layers and masks in the documentation for more information.

Color **debug_shape_custom_color** = `Color(0, 0, 0, 1)`

- void **set_debug_shape_custom_color**(value: Color)

- Color **get_debug_shape_custom_color**()

The custom color to use to draw the shape in the editor and at run-time if **Visible Collision Shapes** is enabled in the **Debug** menu. This color will be highlighted at run-time if the **RayCast3D** is colliding with something.

If set to `Color(0.0, 0.0, 0.0)` (by default), the color set in ProjectSettings.debug/shapes/collision/shape_color is used.

---

int **debug_shape_thickness** = `2` 🔗

- void **set_debug_shape_thickness**(value: int)
- int **get_debug_shape_thickness**()

If set to `1`, a line is used as the debug shape. Otherwise, a truncated pyramid is drawn to represent the **RayCast3D**. Requires **Visible Collision Shapes** to be enabled in the **Debug** menu for the debug shape to be visible at run-time.

---

bool **enabled** = `true` 🔗

- void **set_enabled**(value: bool)
- bool **is_enabled**()

If `true`, collisions will be reported.

---

bool **exclude_parent** = `true` 🔗

- void **set_exclude_parent_body**(value: bool)
- bool **get_exclude_parent_body**()

If `true`, collisions will be ignored for this RayCast3D's immediate parent.

---

bool **hit_back_faces** = `true` 🔗

- void **set_hit_back_faces**(value: bool)
- bool **is_hit_back_faces_enabled**()

If `true`, the ray will hit back faces with concave polygon shapes with back face enabled or heightmap shapes.

---

bool **hit_from_inside** = `false` 🔗

- void **set_hit_from_inside**(value: bool)
- bool **is_hit_from_inside_enabled**()

If `true`, the ray will detect a hit when starting inside shapes. In this case the collision normal will be `Vector3(0, 0, 0)`. Does not affect shapes with no volume like concave polygon or heightmap.

Vector3 **target_position** = `Vector3(0, -1, 0)` 🔗

- void **set_target_position**(value: Vector3)
- Vector3 **get_target_position**()

The ray's destination point, relative to the RayCast's `position`.

Method Descriptions

void **add_exception**(node: CollisionObject3D) 🔗

Adds a collision exception so the ray does not report collisions with the specified CollisionObject3D node.

---

void **add_exception_rid**(rid: RID)

Adds a collision exception so the ray does not report collisions with the specified RID.

---

void **clear_exceptions**()

Removes all collision exceptions for this ray.

---

void **force_raycast_update**()

Updates the collision information for the ray immediately, without waiting for the next `_physics_process` call. Use this method, for example, when the ray or its parent has changed state.

Note: enabled does not need to be `true` for this to work.

Object **get_collider**() const

Returns the first object that the ray intersects, or `null` if no object is intersecting the ray (i.e. is_colliding returns `false`).

RID **get_collider_rid**() const

Returns the RID of the first object that the ray intersects, or an empty RID if no object is intersecting the ray (i.e. is_colliding returns `false`).

---

int **get_collider_shape**() const

Returns the shape ID of the first object that the ray intersects, or `0` if no object is intersecting the ray (i.e. is_colliding returns `false`).

To get the intersected shape node, for a CollisionObject3D target, use:

GDScript

```
var target = get_collider() # A CollisionObject3D.
var shape_id = get_collider_shape() # The shape index in the
collider.
var owner_id = target.shape_find_owner(shape_id) # The owner ID
in the collider.
var shape = target.shape_owner_get_owner(owner_id)
```

C#

```
var target = (CollisionObject3D)GetCollider(); // A
CollisionObject3D.
var shapeId = GetColliderShape(); // The shape index in the
collider.
var ownerId = target.ShapeFindOwner(shapeId); // The owner ID
in the collider.
var shape = target.ShapeOwnerGetOwner(ownerId);
```

---

int **get_collision_face_index**() const 🔗

Returns the collision object's face index at the collision point, or `-1` if the shape intersecting the ray is not a ConcavePolygonShape3D.

---

bool **get_collision_mask_value**(layer_number: int) const 🔗

Returns whether or not the specified layer of the collision_mask is enabled, given a `layer_number` between 1 and 32.

Vector3 **get_collision_normal**() const 🔗

Returns the normal of the intersecting object's shape at the collision point, or `Vector3(0, 0, 0)` if the ray starts inside the shape and hit_from_inside is `true`.

Note: Check that is_colliding returns `true` before calling this method to ensure the returned normal is valid and up-to-date.

Vector3 **get_collision_point**() const

Returns the collision point at which the ray intersects the closest object, in the global coordinate system. If hit_from_inside is `true` and the ray starts inside of a collision shape, this function will return the origin point of the ray.

Note: Check that is_colliding returns `true` before calling this method to ensure the returned point is valid and up-to-date.

---

bool **is_colliding**() const

Returns whether any object is intersecting with the ray's vector (considering the vector length).

---

void **remove_exception**(node: CollisionObject3D)

Removes a collision exception so the ray does report collisions with the specified CollisionObject3D node.

---

void **remove_exception_rid**(rid: RID)

Removes a collision exception so the ray does report collisions with the specified RID.

---

void **set_collision_mask_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the collision_mask, given a `layer_number` between 1 and 32.

ReferenceRect

Inherits: Control **<** CanvasItem **<** Node **<** Object

A rectangle hint for designing UIs.

Description

A rectangle box that displays only a colored border around its rectangle. It is used to visualize the extents of a Control.

Properties

Color	border_color	`Color(1, 0, 0, 1)`
float	border_width	`1.0`
bool	editor_only	`true`

Property Descriptions

Color **border_color** = `Color(1, 0, 0, 1)` 🔗

- void **set_border_color**(value: Color)
- Color **get_border_color**()

Sets the border color of the **ReferenceRect**.

---

float **border_width** = `1.0`

- void **set_border_width**(value: float)
- float **get_border_width**()

Sets the border width of the **ReferenceRect**. The border grows both inwards and outwards with respect to the rectangle box.

---

bool **editor_only** = `true`

- void **set_editor_only**(value: bool)
- bool **get_editor_only**()

If `true`, the **ReferenceRect** will only be visible while in editor. Otherwise, **ReferenceRect** will be visible in the running project.

ReflectionProbe

Inherits: VisualInstance3D **<** Node3D **<** Node **<** Object

Captures its surroundings to create fast, accurate reflections from a given point.

Description

Captures its surroundings as a cubemap, and stores versions of it with increasing levels of blur to simulate different material roughnesses.

The **ReflectionProbe** is used to create high-quality reflections at a low performance cost (when update_mode is UPDATE_ONCE). **ReflectionProbe**s can be blended together and with the rest of the scene smoothly. **ReflectionProbe**s can also be combined with VoxelGI, SDFGI (Environment.sdfgi_enabled) and screen-space reflections (Environment.ssr_enabled) to get more accurate reflections in specific areas. **ReflectionProbe**s render all objects within their cull_mask, so updating them can be quite expensive. It is best to update them once with the important static objects and then leave them as-is.

Note: Unlike VoxelGI and SDFGI, **ReflectionProbe**s only source their environment from a WorldEnvironment node. If you specify an Environment resource within a Camera3D node, it will be ignored by the **ReflectionProbe**. This can lead to incorrect lighting within the **ReflectionProbe**.

Note: Reflection probes are only supported in the Forward+ and Mobile rendering methods, not Compatibility. When using the Mobile rendering method, only 8 reflection probes can be displayed on each mesh resource. Attempting to display more than 8 reflection probes on a single mesh resource will result in reflection probes flickering in and out as the camera moves.

Note: When using the Mobile rendering method, reflection probes will only correctly affect meshes whose visibility AABB intersects with the reflection probe's AABB. If using a shader to deform the mesh in a way that makes it go outside its AABB, GeometryInstance3D.extra_cull_margin must be increased on the mesh. Otherwise, the reflection probe may not be visible on the mesh.

Tutorials

- Reflection probes

Properties

Color	ambient_color	`Color(0, 0, 0, 1)`
float	ambient_color_energy	`1.0`
AmbientMode	ambient_mode	`1`
bool	box_projection	`false`
int	cull_mask	`1048575`
bool	enable_shadows	`false`

float	intensity	`1.0`
bool	interior	`false`
float	max_distance	`0.0`
float	mesh_lod_threshold	`1.0`
Vector3	origin_offset	`Vector3(0, 0, 0)`
int	reflection_mask	`1048575`
Vector3	size	`Vector3(20, 20, 20)`
UpdateMode	update_mode	`0`

Enumerations

enum **UpdateMode**: 🔗

UpdateMode **UPDATE_ONCE** = `0`

Update the probe once on the next frame (recommended for most objects). The corresponding radiance map will be generated over the following six frames. This takes more time to update than UPDATE_ALWAYS, but it has a lower performance cost and can result in higher-quality reflections. The ReflectionProbe is updated when its transform changes, but not when nearby geometry changes. You can force a **ReflectionProbe** update by moving the **ReflectionProbe** slightly in any direction.

UpdateMode **UPDATE_ALWAYS** = `1`

Update the probe every frame. This provides better results for fast-moving dynamic objects (such as cars). However, it has a significant performance cost. Due to the cost, it's recommended to only use one ReflectionProbe with UPDATE_ALWAYS at most per scene. For all other use cases, use UPDATE_ONCE.

enum **AmbientMode**: 🔗

AmbientMode **AMBIENT_DISABLED** = `0`

Do not apply any ambient lighting inside the **ReflectionProbe**'s box defined by its size.

AmbientMode **AMBIENT_ENVIRONMENT** = `1`

Apply automatically-sourced environment lighting inside the **ReflectionProbe**'s box defined by its size.

AmbientMode **AMBIENT_COLOR** = `2`

Apply custom ambient lighting inside the **ReflectionProbe**'s box defined by its size. See ambient_color and ambient_color_energy.

Property Descriptions

Color **ambient_color** = `Color(0, 0, 0, 1)` 🔗

- void **set_ambient_color**(value: Color)
- Color **get_ambient_color**()

The custom ambient color to use within the **ReflectionProbe**'s box defined by its size. Only effective if ambient_mode is AMBIENT_COLOR.

---

float **ambient_color_energy** = `1.0` 🔗

- void **set_ambient_color_energy**(value: float)
- float **get_ambient_color_energy**()

The custom ambient color energy to use within the **ReflectionProbe**'s box defined by its size. Only effective if ambient_mode is AMBIENT_COLOR.

AmbientMode **ambient_mode** = `1` 🔗

- void **set_ambient_mode**(value: AmbientMode)
- AmbientMode **get_ambient_mode**()

The ambient color to use within the **ReflectionProbe**'s box defined by its size. The ambient color will smoothly blend with other **ReflectionProbe**s and the rest of the scene (outside the **ReflectionProbe**'s box defined by its size).

---

bool **box_projection** = `false` 🔗

- void **set_enable_box_projection**(value: bool)
- bool **is_box_projection_enabled**()

If `true`, enables box projection. This makes reflections look more correct in rectangle-shaped rooms by offsetting the reflection center depending on the camera's location.

Note: To better fit rectangle-shaped rooms that are not aligned to the grid, you can rotate the **ReflectionProbe** node.

---

int **cull_mask** = `1048575` 🔗

- void **set_cull_mask**(value: int)
- int **get_cull_mask**()

Sets the cull mask which determines what objects are drawn by this probe. Every VisualInstance3D with a layer included in this cull mask will be rendered by the probe. It is best to only include large objects which are likely to take up a lot of space in the reflection in order to save on rendering cost.

This can also be used to prevent an object from reflecting upon itself (for instance, a **ReflectionProbe** centered on a vehicle).

---

bool **enable_shadows** = `false` 🔗

- void **set_enable_shadows**(value: bool)
- bool **are_shadows_enabled**()

If `true`, computes shadows in the reflection probe. This makes the reflection probe slower to render; you may want to disable this if using the UPDATE_ALWAYS update_mode.

---

float **intensity** = `1.0` 🔗

- void **set_intensity**(value: float)
- float **get_intensity**()

Defines the reflection intensity. Intensity modulates the strength of the reflection.

---

bool **interior** = `false`

- void **set_as_interior**(value: bool)
- bool **is_set_as_interior**()

If `true`, reflections will ignore sky contribution.

---

float **max_distance** = `0.0`

- void **set_max_distance**(value: float)
- float **get_max_distance**()

The maximum distance away from the **ReflectionProbe** an object can be before it is culled. Decrease this to improve performance, especially when using the UPDATE_ALWAYS update_mode.

Note: The maximum reflection distance is always at least equal to the probe's extents. This means that decreasing max_distance will not always cull objects from reflections, especially if the reflection probe's box defined by its size is already large.

---

float **mesh_lod_threshold** = `1.0`

- void **set_mesh_lod_threshold**(value: float)
- float **get_mesh_lod_threshold**()

The automatic LOD bias to use for meshes rendered within the **ReflectionProbe** (this is analog to Viewport.mesh_lod_threshold). Higher values will use less detailed versions of meshes that have LOD variations generated. If set to `0.0`, automatic LOD is disabled. Increase mesh_lod_threshold to improve performance at the cost of geometry detail, especially when using the UPDATE_ALWAYS update_mode.

Note: mesh_lod_threshold does not affect GeometryInstance3D visibility ranges (also known as "manual" LOD or hierarchical LOD).

Vector3 **origin_offset** = `Vector3(0, 0, 0)` 🔗

- void **set_origin_offset**(value: Vector3)
- Vector3 **get_origin_offset**()

Sets the origin offset to be used when this **ReflectionProbe** is in box_projection mode. This can be set to a non-zero value to ensure a reflection fits a rectangle-shaped room, while reducing the number of objects that "get in the way" of the reflection.

---

int **reflection_mask** = `1048575` 🔗

- void **set_reflection_mask**(value: int)
- int **get_reflection_mask**()

Sets the reflection mask which determines what objects have reflections applied from this probe. Every VisualInstance3D with a layer included in this reflection mask will have reflections applied from this probe. See also cull_mask, which can be used to exclude objects from appearing in the reflection while still making them affected by the **ReflectionProbe**.

Vector3 **size** = `Vector3(20, 20, 20)` 🔗

- void **set_size**(value: Vector3)
- Vector3 **get_size**()

The size of the reflection probe. The larger the size, the more space covered by the probe, which will lower the perceived resolution. It is best to keep the size only as large as you need it.

Note: To better fit areas that are not aligned to the grid, you can rotate the **ReflectionProbe** node.

UpdateMode **update_mode** = `0` 🔗

- void **set_update_mode**(value: UpdateMode)
- UpdateMode **get_update_mode**()

Sets how frequently the **ReflectionProbe** is updated. Can be UPDATE_ONCE or UPDATE_ALWAYS.

RemoteTransform2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

RemoteTransform2D pushes its own Transform2D to another Node2D derived node in the scene.

Description

RemoteTransform2D pushes its own Transform2D to another Node2D derived node (called the remote node) in the scene.

It can be set to update another node's position, rotation and/or scale. It can use either global or local coordinates.

Properties

NodePath	remote_path	`NodePath("")`
bool	update_position	`true`
bool	update_rotation	`true`
bool	update_scale	`true`
bool	use_global_coordinates	`true`

Methods

void **force_update_cache**()

Property Descriptions

NodePath **remote_path** = `NodePath("")`

- void **set_remote_node**(value: NodePath)
- NodePath **get_remote_node**()

The NodePath to the remote node, relative to the RemoteTransform2D's position in the scene.

---

bool **update_position** = `true`

- void **set_update_position**(value: bool)
- bool **get_update_position**()

If `true`, the remote node's position is updated.

---

bool **update_rotation** = `true`

- void **set_update_rotation**(value: bool)
- bool **get_update_rotation**()

If `true`, the remote node's rotation is updated.

---

bool **update_scale** = `true`

- void **set_update_scale**(value: bool)
- bool **get_update_scale**()

If `true`, the remote node's scale is updated.

---

bool **use_global_coordinates** = `true` 🔗

- void **set_use_global_coordinates**(value: bool)
- bool **get_use_global_coordinates**()

If `true`, global coordinates are used. If `false`, local coordinates are used.

Method Descriptions

void **force_update_cache**() 🔗

RemoteTransform2D caches the remote node. It may not notice if the remote node disappears; force_update_cache forces it to update the cache again.

RemoteTransform3D

Inherits: Node3D **<** Node **<** Object

RemoteTransform3D pushes its own Transform3D to another Node3D derived Node in the scene.

Description

RemoteTransform3D pushes its own Transform3D to another Node3D derived Node (called the remote node) in the scene.

It can be set to update another Node's position, rotation and/or scale. It can use either global or local coordinates.

Properties

NodePath	remote_path	`NodePath("")`
bool	update_position	`true`
bool	update_rotation	`true`
bool	update_scale	`true`
bool	use_global_coordinates	`true`

Methods

void force_update_cache()

Property Descriptions

NodePath **remote_path** = `NodePath("")` 🔗

- void **set_remote_node**(value: NodePath)
- NodePath **get_remote_node**()

The NodePath to the remote node, relative to the RemoteTransform3D's position in the scene.

---

bool **update_position** = `true` 🔗

- void **set_update_position**(value: bool)
- bool **get_update_position**()

If `true`, the remote node's position is updated.

---

bool **update_rotation** = `true` 🔗

- void **set_update_rotation**(value: bool)
- bool **get_update_rotation**()

If `true`, the remote node's rotation is updated.

---

bool **update_scale** = `true` 🔗

- void **set_update_scale**(value: bool)
- bool **get_update_scale**()

If `true`, the remote node's scale is updated.

bool **use_global_coordinates** = `true` 🔗

- void **set_use_global_coordinates**(value: bool)
- bool **get_use_global_coordinates**()

If `true`, global coordinates are used. If `false`, local coordinates are used.

Method Descriptions

void **force_update_cache**() 🔗

RemoteTransform3D caches the remote node. It may not notice if the remote node disappears; force_update_cache forces it to update the cache again.

ResourcePreloader

Inherits: Node **<** Object

A node used to preload sub-resources inside a scene.

Description

This node is used to preload sub-resources inside a scene, so when the scene is loaded, all the resources are ready to use and can be retrieved from the preloader. You can add the resources using the ResourcePreloader tab when the node is selected.

GDScript has a simplified @GDScript.preload built-in method which can be used in most situations, leaving the use of **ResourcePreloader** for more advanced scenarios.

Methods

void	add_resource(name: StringName, resource: Resource)
Resource	get_resource(name: StringName) const
PackedStringArray	get_resource_list() const
bool	has_resource(name: StringName) const

void	remove_resource(name: StringName)
void	rename_resource(name: StringName, newname: StringName)

Method Descriptions

void **add_resource**(name: StringName, resource: Resource)

Adds a resource to the preloader with the given `name`. If a resource with the given `name` already exists, the new resource will be renamed to "`name` N" where N is an incrementing number starting from 2.

Resource **get_resource**(name: StringName) const

Returns the resource associated to `name`.

PackedStringArray **get_resource_list**() const

Returns the list of resources inside the preloader.

bool **has_resource**(name: StringName) const

Returns `true` if the preloader contains a resource associated to `name`.

void **remove_resource**(name: StringName)

Removes the resource associated to `name` from the preloader.

---

void **rename_resource**(name: StringName, newname: StringName) 🔗

Renames a resource inside the preloader from `name` to `newname`.

RichTextLabel

Inherits: Control < CanvasItem < Node < Object

A control for displaying text that can contain different font styles, images, and basic formatting.

Description

A control for displaying text that can contain custom fonts, images, and basic formatting. **RichTextLabel** manages these as an internal tag stack. It also adapts itself to given width/heights.

Note: Assignments to text clear the tag stack and reconstruct it from the property's contents. Any edits made to text will erase previous edits made from other manual sources such as append_text and the `push_*` / pop methods.

Note: RichTextLabel doesn't support entangled BBCode tags. For example, instead of using `[b]bold[i]bold italic[/b]italic[/i]`, use `[b]bold[i]bold italic[/i][/b][i]italic[/i]`.

Note: `push_*/pop_*` functions won't affect BBCode.

Note: Unlike Label, **RichTextLabel** doesn't have a *property* to horizontally align text to the center. Instead, enable bbcode_enabled and surround the text in a `[center]` tag as follows: `[center]Example[/center]`. There is currently no built-in way to vertically align text either, but this can be emulated by relying on anchors/containers and the fit_content property.

Tutorials

- BBCode in RichTextLabel
- Rich Text Label with BBCode Demo [https://godotengine.org/asset-library/asset/2774]

- Operating System Testing Demo [https://godotengine.org/asset-library/asset/2789]

Properties

AutowrapMode	autowrap_mode	`3`
bool	bbcode_enabled	`false`
bool	clip_contents	`true` (overrides Control)
bool	context_menu_enabled	`false`
Array	custom_effects	`[]`
bool	deselect_on_focus_loss_enabled	`true`
bool	drag_and_drop_selection_enabled	`true`
bool	fit_content	`false`
bool	hint_underlined	`true`
String	language	`""`
bool	meta_underlined	`true`

int	progress_bar_delay	`1000`
bool	scroll_active	`true`
bool	scroll_following	`false`
bool	selection_enabled	`false`
bool	shortcut_keys_enabled	`true`
StructuredTextParser	structured_text_bidi_override	`0`
Array	structured_text_bidi_override_options	`[]`
int	tab_size	`4`
String	text	`""`
TextDirection	text_direction	`0`
bool	threaded	`false`
int	visible_characters	`-1`
VisibleCharactersBehavior	visible_characters_behavior	`0`
float	visible_ratio	`1.0`

Methods

void	add_image(image: Texture2D, width: int = 0, height: int = 0, color: Color = Color(1, 1, 1, 1), inline_align: InlineAlignment = 5, region: Rect2 = Rect2(0, 0, 0, 0), key: Variant = null, pad: bool = false, tooltip: String = "", size_in_percent: bool = false)
void	add_text(text: String)
void	append_text(bbcode: String)
void	clear()
void	deselect()
int	get_character_line(character: int)
int	get_character_paragraph(character: int)
int	get_content_height() const
int	get_content_width() const
int	get_line_count() const
float	get_line_offset(line: int)
PopupMenu	get_menu() const

int	get_paragraph_count() const
float	get_paragraph_offset(paragraph: int)
String	get_parsed_text() const
String	get_selected_text() const
int	get_selection_from() const
int	get_selection_to() const
int	get_total_character_count() const
VScrollBar	get_v_scroll_bar()
int	get_visible_line_count() const
int	get_visible_paragraph_count() const
void	install_effect(effect: Variant)
bool	invalidate_paragraph(paragraph: int)
bool	is_menu_visible() const
bool	is_ready() const

void	menu_option(option: int)
void	newline()
void	parse_bbcode(bbcode: String)
Dictionary	parse_expressions_for_values(expressions: PackedStringArray)
void	pop()
void	pop_all()
void	pop_context()
void	push_bgcolor(bgcolor: Color)
void	push_bold()
void	push_bold_italics()
void	push_cell()
void	push_color(color: Color)
void	push_context()
void	push_customfx(effect: RichTextEffect, env: Dictionary)

void	push_dropcap(string: String, font: Font, size: int, dropcap_margins: Rect2 = Rect2(0, 0, 0, 0), color: Color = Color(1, 1, 1, 1), outline_size: int = 0, outline_color: Color = Color(0, 0, 0, 0))
void	push_fgcolor(fgcolor: Color)
void	push_font(font: Font, font_size: int = 0)
void	push_font_size(font_size: int)
void	push_hint(description: String)
void	push_indent(level: int)
void	push_italics()
void	push_language(language: String)
void	push_list(level: int, type: ListType, capitalize: bool, bullet: String = "•")
void	push_meta(data: Variant, underline_mode: MetaUnderline = 1)
void	push_mono()
void	push_normal()

void	push_outline_color(color: Color)
void	push_outline_size(outline_size: int)
void	push_paragraph(alignment: HorizontalAlignment, base_direction: TextDirection = 0, language: String = "", st_parser: StructuredTextParser = 0, justification_flags: BitField[JustificationFlag] = 163, tab_stops: PackedFloat32Array = PackedFloat32Array())
void	push_strikethrough()
void	push_table(columns: int, inline_align: InlineAlignment = 0, align_to_row: int = -1)
void	push_underline()
bool	remove_paragraph(paragraph: int, no_invalidate: bool = false)
void	scroll_to_line(line: int)
void	scroll_to_paragraph(paragraph: int)
void	scroll_to_selection()
void	select_all()
void	set_cell_border_color(color: Color)

void	set_cell_padding(padding: Rect2)
void	set_cell_row_background_color(odd_row_bg: Color, even_row_bg: Color)
void	set_cell_size_override(min_size: Vector2, max_size: Vector2)
void	set_table_column_expand(column: int, expand: bool, ratio: int = 1)
void	update_image(key: Variant, mask: BitField[ImageUpdateMask], image: Texture2D, width: int = 0, height: int = 0, color: Color = Color(1, 1, 1, 1), inline_align: InlineAlignment = 5, region: Rect2 = Rect2(0, 0, 0, 0), pad: bool = false, tooltip: String = "", size_in_percent: bool = false)

Theme Properties

Color	default_color	`Color(1, 1, 1, 1)`
Color	font_outline_color	`Color(0, 0, 0, 1)`
Color	font_selected_color	`Color(0, 0, 0, 0)`
Color	font_shadow_color	`Color(0, 0, 0, 0)`
Color	selection_color	`Color(0.1, 0.1, 1, 0.8)`

Color	table_border	`Color(0, 0, 0, 0)`
Color	table_even_row_bg	`Color(0, 0, 0, 0)`
Color	table_odd_row_bg	`Color(0, 0, 0, 0)`
int	line_separation	`0`
int	outline_size	`0`
int	shadow_offset_x	`1`
int	shadow_offset_y	`1`
int	shadow_outline_size	`1`
int	table_h_separation	`3`
int	table_v_separation	`3`
int	text_highlight_h_padding	`3`
int	text_highlight_v_padding	`3`
Font	bold_font	
Font	bold_italics_font	

Font	italics_font
Font	mono_font
Font	normal_font
int	bold_font_size
int	bold_italics_font_size
int	italics_font_size
int	mono_font_size
int	normal_font_size
StyleBox	focus
StyleBox	normal

Signals

finished()

Triggered when the document is fully loaded.

---

meta_clicked(meta: Variant)

Triggered when the user clicks on content between meta (URL) tags. If the meta is defined in BBCode, e.g. `[url={"key": "value"}]Text[/url]`, then the parameter for this signal will always be a String type. If a particular type or an object is desired, the push_meta method must be used to manually insert the data into the tag stack. Alternatively, you can convert the String input to the desired type based on its contents (such as calling JSON.parse on it).

For example, the following method can be connected to meta_clicked to open clicked URLs using the user's default web browser:

GDScript

```
# This assumes RichTextLabel's `meta_clicked` signal was connected to
# the function below using the signal connection dialog.
func _richtextlabel_on_meta_clicked(meta):
    # `meta` is of Variant type, so convert it to a String to avoid script errors at run-time.
    OS.shell_open(str(meta))
```

meta_hover_ended(meta: Variant)

Triggers when the mouse exits a meta tag.

meta_hover_started(meta: Variant)

Triggers when the mouse enters a meta tag.

Enumerations

enum **ListType**:

ListType **LIST_NUMBERS** = `0`

Each list item has a number marker.

ListType **LIST_LETTERS** = `1`

Each list item has a letter marker.

ListType **LIST_ROMAN** = `2`

Each list item has a roman number marker.

ListType **LIST_DOTS** = `3`

Each list item has a filled circle marker.

---

enum **MenuItems**: 🔗

MenuItems **MENU_COPY** = `0`

Copies the selected text.

MenuItems **MENU_SELECT_ALL** = `1`

Selects the whole **RichTextLabel** text.

MenuItems **MENU_MAX** = `2`

Represents the size of the MenuItems enum.

---

enum **MetaUnderline**: 🔗

MetaUnderline **META_UNDERLINE_NEVER** = `0`

Meta tag does not display an underline, even if meta_underlined is `true`.

MetaUnderline **META_UNDERLINE_ALWAYS** = `1`

If meta_underlined is `true`, meta tag always display an underline.

MetaUnderline **META_UNDERLINE_ON_HOVER** = `2`

If meta_underlined is `true`, meta tag display an underline when the mouse cursor is over it.

---

flags **ImageUpdateMask**: 🔗

ImageUpdateMask **UPDATE_TEXTURE** = `1`

If this bit is set, update_image changes image texture.

ImageUpdateMask **UPDATE_SIZE** = `2`

If this bit is set, update_image changes image size.

ImageUpdateMask **UPDATE_COLOR** = `4`

If this bit is set, update_image changes image color.

ImageUpdateMask **UPDATE_ALIGNMENT** = `8`

If this bit is set, update_image changes image inline alignment.

ImageUpdateMask **UPDATE_REGION** = `16`

If this bit is set, update_image changes image texture region.

ImageUpdateMask **UPDATE_PAD** = `32`

If this bit is set, update_image changes image padding.

ImageUpdateMask **UPDATE_TOOLTIP** = `64`

If this bit is set, update_image changes image tooltip.

ImageUpdateMask **UPDATE_WIDTH_IN_PERCENT** = `128`

If this bit is set, update_image changes image width from/to percents.

Property Descriptions

AutowrapMode **autowrap_mode** = `3` 🔗

- void **set_autowrap_mode**(value: AutowrapMode)
- AutowrapMode **get_autowrap_mode**()

If set to something other than TextServer.AUTOWRAP_OFF, the text gets wrapped inside the node's bounding rectangle. To see how each mode behaves, see AutowrapMode.

bool **bbcode_enabled** = `false`

- void **set_use_bbcode**(value: bool)
- bool **is_using_bbcode**()

If `true`, the label uses BBCode formatting.

Note: This only affects the contents of text, not the tag stack.

bool **context_menu_enabled** = `false`

- void **set_context_menu_enabled**(value: bool)
- bool **is_context_menu_enabled**()

If `true`, a right-click displays the context menu.

Array **custom_effects** = `[]`

- void **set_effects**(value: Array)
- Array **get_effects**()

The currently installed custom effects. This is an array of RichTextEffects.

To add a custom effect, it's more convenient to use install_effect.

bool **deselect_on_focus_loss_enabled** = `true`

- void **set_deselect_on_focus_loss_enabled**(value: bool)
- bool **is_deselect_on_focus_loss_enabled**()

If `true`, the selected text will be deselected when focus is lost.

bool **drag_and_drop_selection_enabled** = `true`

- void **set_drag_and_drop_selection_enabled**(value: bool)
- bool **is_drag_and_drop_selection_enabled**()

If `true`, allow drag and drop of selected text.

---

bool **fit_content** = `false`

- void **set_fit_content**(value: bool)
- bool **is_fit_content_enabled**()

If `true`, the label's minimum size will be automatically updated to fit its content, matching the behavior of Label.

---

bool **hint_underlined** = `true`

- void **set_hint_underline**(value: bool)
- bool **is_hint_underlined**()

If `true`, the label underlines hint tags such as `[hint=description]{text}[/hint]`.

String **language** = `""`

- void **set_language**(value: String)
- String **get_language**()

Language code used for line-breaking and text shaping algorithms, if left empty current locale is used instead.

---

bool **meta_underlined** = `true`

- void **set_meta_underline**(value: bool)
- bool **is_meta_underlined**()

If `true`, the label underlines meta tags such as `[url]{text}[/url]`. These tags can call a function when clicked if meta_clicked is connected to a function.

---

int **progress_bar_delay** = `1000` 🔗

- void **set_progress_bar_delay**(value: int)
- int **get_progress_bar_delay**()

The delay after which the loading progress bar is displayed, in milliseconds. Set to `-1` to disable progress bar entirely.

Note: Progress bar is displayed only if threaded is enabled.

---

bool **scroll_active** = `true` 🔗

- void **set_scroll_active**(value: bool)
- bool **is_scroll_active**()

If `true`, the scrollbar is visible. Setting this to `false` does not block scrolling completely. See scroll_to_line.

---

bool **scroll_following** = `false` 🔗

- void **set_scroll_follow**(value: bool)
- bool **is_scroll_following**()

If `true`, the window scrolls down to display new content automatically.

---

bool **selection_enabled** = `false` 🔗

- void **set_selection_enabled**(value: bool)
- bool **is_selection_enabled**()

If `true`, the label allows text selection.

---

bool **shortcut_keys_enabled** = `true`

- void **set_shortcut_keys_enabled**(value: bool)
- bool **is_shortcut_keys_enabled**()

If `true`, shortcut keys for context menu items are enabled, even if the context menu is disabled.

StructuredTextParser **structured_text_bidi_override** = `0`

- void **set_structured_text_bidi_override**(value: StructuredTextParser)
- StructuredTextParser **get_structured_text_bidi_override**()

Set BiDi algorithm override for the structured text.

Array **structured_text_bidi_override_options** = `[]`

- void **set_structured_text_bidi_override_options**(value: Array)
- Array **get_structured_text_bidi_override_options**()

Set additional options for BiDi override.

---

int **tab_size** = `4`

- void **set_tab_size**(value: int)
- int **get_tab_size**()

The number of spaces associated with a single tab length. Does not affect `\t` in text tags, only indent tags.

String **text** = `""`

- void **set_text**(value: String)
- String **get_text**()

The label's text in BBCode format. Is not representative of manual modifications to the internal tag stack. Erases changes made by other methods when edited.

Note: If bbcode_enabled is `true`, it is unadvised to use the `+=` operator with text (e.g. `text += "some string"`) as it replaces the whole text and can cause slowdowns. It will also erase all BBCode that was added to stack using `push_*` methods. Use append_text for adding text instead, unless you absolutely need to close a tag that was opened in an earlier method call.

TextDirection **text_direction** = `0` 🔗

- void **set_text_direction**(value: TextDirection)
- TextDirection **get_text_direction**()

Base text writing direction.

---

bool **threaded** = `false` 🔗

- void **set_threaded**(value: bool)
- bool **is_threaded**()

If `true`, text processing is done in a background thread.

---

int **visible_characters** = `-1` 🔗

- void **set_visible_characters**(value: int)
- int **get_visible_characters**()

The number of characters to display. If set to `-1`, all characters are displayed. This can be useful when animating the text appearing in a dialog box.

Note: Setting this property updates visible_ratio accordingly.

VisibleCharactersBehavior **visible_characters_behavior** = `0` 🔗

- void **set_visible_characters_behavior**(value: VisibleCharactersBehavior)
- VisibleCharactersBehavior **get_visible_characters_behavior**()

Sets the clipping behavior when visible_characters or visible_ratio is set. See VisibleCharactersBehavior for more info.

---

float **visible_ratio** = `1.0` 🔗

- void **set_visible_ratio**(value: float)
- float **get_visible_ratio**()

The fraction of characters to display, relative to the total number of characters (see get_total_character_count). If set to `1.0`, all characters are displayed. If set to `0.5`, only half of the characters will be displayed. This can be useful when animating the text appearing in a dialog box.

Note: Setting this property updates visible_characters accordingly.

Method Descriptions

void **add_image**(image: Texture2D, width: int = 0, height: int = 0, color: Color = Color(1, 1, 1, 1), inline_align: InlineAlignment = 5, region: Rect2 = Rect2(0, 0, 0, 0), key: Variant = null, pad: bool = false, tooltip: String = "", size_in_percent: bool = false) 🔗

Adds an image's opening and closing tags to the tag stack, optionally providing a `width` and `height` to resize the image, a `color` to tint the image and a `region` to only use parts of the image.

If `width` or `height` is set to 0, the image size will be adjusted in order to keep the original aspect ratio.

If `width` and `height` are not set, but `region` is, the region's rect will be used.

`key` is an optional identifier, that can be used to modify the image via update_image.

If `pad` is set, and the image is smaller than the size specified by `width` and `height`, the image padding is added to match the size instead of upscaling.

If `size_in_percent` is set, `width` and `height` values are percentages of the control width instead of pixels.

---

void **add_text**(text: String) 🔗

Adds raw non-BBCode-parsed text to the tag stack.

---

void **append_text**(bbcode: String) 🔗

Parses `bbcode` and adds tags to the tag stack as needed.

Note: Using this method, you can't close a tag that was opened in a previous append_text call. This is done to improve performance, especially when updating large RichTextLabels since rebuilding the whole BBCode every time would be slower. If you absolutely need to close a tag in a future method call, append the text instead of using append_text.

---

void **clear**() 🔗

Clears the tag stack, causing the label to display nothing.

Note: This method does not affect text, and its contents will show again if the label is redrawn. However, setting text to an empty String also clears the stack.

---

void **deselect**() 🔗

Clears the current selection.

---

int **get_character_line**(character: int) 🔗

Returns the line number of the character position provided. Line and character numbers are both zero-indexed.

Note: If threaded is enabled, this method returns a value for the loaded part of the document. Use is_ready or finished to determine whether document is fully loaded.

---

int **get_character_paragraph**(character: int)

Returns the paragraph number of the character position provided. Paragraph and character numbers are both zero-indexed.

Note: If threaded is enabled, this method returns a value for the loaded part of the document. Use is_ready or finished to determine whether document is fully loaded.

---

int **get_content_height**() const

Returns the height of the content.

Note: If threaded is enabled, this method returns a value for the loaded part of the document. Use is_ready or finished to determine whether document is fully loaded.

---

int **get_content_width**() const

Returns the width of the content.

Note: If threaded is enabled, this method returns a value for the loaded part of the document. Use is_ready or finished to determine whether document is fully loaded.

---

int **get_line_count**() const

Returns the total number of lines in the text. Wrapped text is counted as multiple lines.

Note: If threaded is enabled, this method returns a value for the loaded part of the document. Use is_ready or finished to determine whether document is fully loaded.

---

float **get_line_offset**(line: int) 🔗

Returns the vertical offset of the line found at the provided index.

Note: If threaded is enabled, this method returns a value for the loaded part of the document. Use is_ready or finished to determine whether document is fully loaded.

PopupMenu **get_menu**() const 🔗

Returns the PopupMenu of this **RichTextLabel**. By default, this menu is displayed when right-clicking on the **RichTextLabel**.

You can add custom menu items or remove standard ones. Make sure your IDs don't conflict with the standard ones (see MenuItems). For example:

GDScript

```
func _ready():
    var menu = get_menu()
    # Remove "Select All" item.
    menu.remove_item(MENU_SELECT_ALL)
    # Add custom items.
    menu.add_separator()
    menu.add_item("Duplicate Text", MENU_MAX + 1)
    # Connect callback.
    menu.id_pressed.connect(_on_item_pressed)

func _on_item_pressed(id):
    if id == MENU_MAX + 1:
        add_text("\n" + get_parsed_text())
```

C#

```
public override void _Ready()
{
    var menu = GetMenu();
    // Remove "Select All" item.
    menu.RemoveItem(RichTextLabel.MenuItems.SelectAll);
    // Add custom items.
    menu.AddSeparator();
    menu.AddItem("Duplicate Text", RichTextLabel.MenuItems.Max + 1);
    // Add event handler.
    menu.IdPressed += OnItemPressed;
}

public void OnItemPressed(int id)
{
    if (id == TextEdit.MenuItems.Max + 1)
    {
        AddText("\n" + GetParsedText());
    }
}
```

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their Window.visible property.

---

int **get_paragraph_count**() const

Returns the total number of paragraphs (newlines or `p` tags in the tag stack's text tags). Considers wrapped text as one paragraph.

---

float **get_paragraph_offset**(paragraph: int)

Returns the vertical offset of the paragraph found at the provided index.

Note: If threaded is enabled, this method returns a value for the loaded part of the document. Use is_ready or finished to determine whether document is fully loaded.

String **get_parsed_text**() const

Returns the text without BBCode mark-up.

String **get_selected_text**() const

Returns the current selection text. Does not include BBCodes.

---

int **get_selection_from**() const

Returns the current selection first character index if a selection is active, `-1` otherwise. Does not include BBCodes.

---

int **get_selection_to**() const 🔗

Returns the current selection last character index if a selection is active, `-1` otherwise. Does not include BBCodes.

---

int **get_total_character_count**() const 🔗

Returns the total number of characters from text tags. Does not include BBCodes.

VScrollBar **get_v_scroll_bar**() 🔗

Returns the vertical scrollbar.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their CanvasItem.visible property.

---

int **get_visible_line_count**() const 🔗

Returns the number of visible lines.

Note: If threaded is enabled, this method returns a value for the loaded part of the document. Use is_ready or finished to determine whether document is fully loaded.

---

int **get_visible_paragraph_count**() const 🔗

Returns the number of visible paragraphs. A paragraph is considered visible if at least one of its lines is visible.

Note: If threaded is enabled, this method returns a value for the loaded part of the document. Use is_ready or finished to determine whether document is fully loaded.

---

void **install_effect**(effect: Variant) 🔗

Installs a custom effect. This can also be done in the RichTextLabel inspector using the custom_effects property. `effect` should be a valid RichTextEffect.

Example RichTextEffect:

```
# effect.gd
class_name MyCustomEffect
extends RichTextEffect

var bbcode = "my_custom_effect"

# ...
```

Registering the above effect in RichTextLabel from script:

```
# rich_text_label.gd
extends RichTextLabel

func _ready():
    install_effect(MyCustomEffect.new())

    # Alternatively, if not using `class_name` in the script that
extends RichTextEffect:
    install_effect(preload("res://effect.gd").new())
```

---

bool **invalidate_paragraph**(paragraph: int) 🔗

Invalidates `paragraph` and all subsequent paragraphs cache.

---

bool **is_menu_visible**() const 🔗

Returns whether the menu is visible. Use this instead of `get_menu().visible` to improve performance (so the creation of the menu is avoided).

---

bool **is_ready**() const 🔗

If threaded is enabled, returns `true` if the background thread has finished text processing, otherwise always return `true`.

---

void **menu_option**(option: int) 🔗

Executes a given action as defined in the MenuItems enum.

---

void **newline**() 🔗

Adds a newline tag to the tag stack.

---

void **parse_bbcode**(bbcode: String) 🔗

The assignment version of append_text. Clears the tag stack and inserts the new content.

Dictionary **parse_expressions_for_values**(expressions: PackedStringArray) 🔗

Parses BBCode parameter `expressions` into a dictionary.

---

void **pop**() 🔗

Terminates the current tag. Use after `push_*` methods to close BBCodes manually. Does not need to follow `add_*` methods.

---

void **pop_all**() 🔗

Terminates all tags opened by `push_*` methods.

---

void **pop_context**() 🔗

Terminates tags opened after the last push_context call (including context marker), or all tags if there's no context marker on the stack.

---

void **push_bgcolor**(bgcolor: Color)

Adds a `[bgcolor]` tag to the tag stack.

---

void **push_bold**()

Adds a `[font]` tag with a bold font to the tag stack. This is the same as adding a `[b]` tag if not currently in a `[i]` tag.

---

void **push_bold_italics**()

Adds a `[font]` tag with a bold italics font to the tag stack.

---

void **push_cell**()

Adds a `[cell]` tag to the tag stack. Must be inside a `[table]` tag. See push_table for details. Use set_table_column_expand to set column expansion ratio, set_cell_border_color to set cell border, set_cell_row_background_color to set cell background, set_cell_size_override to override cell size, and set_cell_padding to set padding.

---

void **push_color**(color: Color)

Adds a `[color]` tag to the tag stack.

---

void **push_context**()

Adds a context marker to the tag stack. See pop_context.

---

void **push_customfx**(effect: RichTextEffect, env: Dictionary)

Adds a custom effect tag to the tag stack. The effect does not need to be in custom_effects. The environment is directly passed to the effect.

---

void **push_dropcap**(string: String, font: Font, size: int, dropcap_margins: Rect2 = Rect2(0, 0, 0, 0), color: Color = Color(1, 1, 1, 1), outline_size: int = 0, outline_color: Color = Color(0, 0, 0, 0))

Adds a `[dropcap]` tag to the tag stack. Drop cap (dropped capital) is a decorative element at the beginning of a paragraph that is larger than the rest of the text.

---

void **push_fgcolor**(fgcolor: Color)

Adds a `[fgcolor]` tag to the tag stack.

---

void **push_font**(font: Font, font_size: int = 0)

Adds a `[font]` tag to the tag stack. Overrides default fonts for its duration.

Passing `0` to `font_size` will use the existing default font size.

---

void **push_font_size**(font_size: int)

Adds a `[font_size]` tag to the tag stack. Overrides default font size for its duration.

---

void **push_hint**(description: String)

Adds a `[hint]` tag to the tag stack. Same as BBCode `[hint=something]{text}[/hint]`.

---

void **push_indent**(level: int)

Adds an `[indent]` tag to the tag stack. Multiplies `level` by current tab_size to determine new margin length.

---

void **push_italics**()

Adds a `[font]` tag with an italics font to the tag stack. This is the same as adding an `[i]` tag if not currently in a `[b]` tag.

---

void **push_language**(language: String)

Adds language code used for text shaping algorithm and Open-Type font features.

---

void **push_list**(level: int, type: ListType, capitalize: bool, bullet: String = "•")

Adds `[ol]` or `[ul]` tag to the tag stack. Multiplies `level` by current tab_size to determine new margin length.

---

void **push_meta**(data: Variant, underline_mode: MetaUnderline = 1)

Adds a meta tag to the tag stack. Similar to the BBCode `[url=something]{text}[/url]`, but supports non-String metadata types.

If meta_underlined is `true`, meta tags display an underline. This behavior can be customized with `underline_mode`.

Note: Meta tags do nothing by default when clicked. To assign behavior when clicked, connect meta_clicked to a function that is called when the meta tag is clicked.

---

void **push_mono**()

Adds a `[font]` tag with a monospace font to the tag stack.

---

void **push_normal**()

Adds a `[font]` tag with a normal font to the tag stack.

---

void **push_outline_color**(color: Color)

Adds a `[outline_color]` tag to the tag stack. Adds text outline for its duration.

---

void **push_outline_size**(outline_size: int)

Adds a `[outline_size]` tag to the tag stack. Overrides default text outline size for its duration.

---

void **push_paragraph**(alignment: HorizontalAlignment, base_direction: TextDirection = 0, language: String = "", st_parser: StructuredTextParser = 0, justification_flags: BitField[JustificationFlag] = 163, tab_stops: PackedFloat32Array = PackedFloat32Array())

Adds a `[p]` tag to the tag stack.

---

void **push_strikethrough**()

Adds a `[s]` tag to the tag stack.

---

void **push_table**(columns: int, inline_align: InlineAlignment = 0, align_to_row: int = -1)

Adds a `[table=columns,inline_align]` tag to the tag stack. Use set_table_column_expand to set column expansion ratio. Use push_cell to add cells.

---

void **push_underline**()

Adds a `[u]` tag to the tag stack.

---

bool **remove_paragraph**(paragraph: int, no_invalidate: bool = false)

Removes a paragraph of content from the label. Returns `true` if the paragraph exists.

The `paragraph` argument is the index of the paragraph to remove, it can take values in the interval `[0, get_paragraph_count() - 1]`.

If `no_invalidate` is set to `true`, cache for the subsequent paragraphs is not invalidated. Use it for faster updates if deleted paragraph is fully self-contained (have no unclosed tags), or this call is part of the complex edit operation and invalidate_paragraph will be called at the end of operation.

void **scroll_to_line**(line: int)

Scrolls the window's top line to match `line`.

void **scroll_to_paragraph**(paragraph: int)

Scrolls the window's top line to match first line of the `paragraph`.

void **scroll_to_selection**()

Scrolls to the beginning of the current selection.

void **select_all**()

Select all the text.

If selection_enabled is `false`, no selection will occur.

void **set_cell_border_color**(color: Color)

Sets color of a table cell border.

void **set_cell_padding**(padding: Rect2)

Sets inner padding of a table cell.

void **set_cell_row_background_color**(odd_row_bg: Color, even_row_bg: Color)

Sets color of a table cell. Separate colors for alternating rows can be specified.

void **set_cell_size_override**(min_size: Vector2, max_size: Vector2)

Sets minimum and maximum size overrides for a table cell.

void **set_table_column_expand**(column: int, expand: bool, ratio: int = 1)

Edits the selected column's expansion options. If `expand` is `true`, the column expands in proportion to its expansion ratio versus the other columns' ratios.

For example, 2 columns with ratios 3 and 4 plus 70 pixels in available width would expand 30 and 40 pixels, respectively.

If `expand` is `false`, the column will not contribute to the total ratio.

void **update_image**(key: Variant, mask: BitField[ImageUpdateMask], image: Texture2D, width: int = 0, height: int = 0, color: Color = Color(1, 1, 1, 1), inline_align: InlineAlignment = 5, region: Rect2 = Rect2(0, 0, 0, 0), pad: bool = false, tooltip: String = "", size_in_percent: bool = false)

Updates the existing images with the key `key`. Only properties specified by `mask` bits are updated. See add_image.

Theme Property Descriptions

Color **default_color** = `Color(1, 1, 1, 1)`

The default text color.

Color **font_outline_color** = `Color(0, 0, 0, 1)`

The default tint of text outline.

Color **font_selected_color** = `Color(0, 0, 0, 0)`

The color of selected text, used when selection_enabled is `true`. If equal to `Color(0, 0, 0, 0)`, it will be ignored.

Color **font_shadow_color** = `Color(0, 0, 0, 0)`

The color of the font's shadow.

Color **selection_color** = `Color(0.1, 0.1, 1, 0.8)`

The color of the selection box.

Color **table_border** = `Color(0, 0, 0, 0)`

The default cell border color.

Color **table_even_row_bg** = `Color(0, 0, 0, 0)`

The default background color for even rows.

Color **table_odd_row_bg** = `Color(0, 0, 0, 0)`

The default background color for odd rows.

---

int **line_separation** = `0`

The vertical space between lines.

---

int **outline_size** = `0`

The size of the text outline.

Note: If using a font with FontFile.multichannel_signed_distance_field enabled, its FontFile.msdf_pixel_range must be set to at least *twice* the value of outline_size for outline rendering to look correct. Otherwise, the outline may appear to be cut off earlier than intended.

---

int **shadow_offset_x** = `1`

The horizontal offset of the font's shadow.

---

int **shadow_offset_y** = `1`

The vertical offset of the font's shadow.

---

int **shadow_outline_size** = `1`

The size of the shadow outline.

---

int **table_h_separation** = `3`

The horizontal separation of elements in a table.

---

int **table_v_separation** = `3`

The vertical separation of elements in a table.

---

int **text_highlight_h_padding** = `3`

The horizontal padding around boxes drawn by the `[fgcolor]` and `[bgcolor]` tags. This does not affect the appearance of text selection.

---

int **text_highlight_v_padding** = `3`

The vertical padding around boxes drawn by the `[fgcolor]` and `[bgcolor]` tags. This does not affect the appearance of text selection.

Font **bold_font**

The font used for bold text.

Font **bold_italics_font**

The font used for bold italics text.

Font **italics_font**

The font used for italics text.

Font **mono_font**

The font used for monospace text.

Font **normal_font**

The default text font.

---

int **bold_font_size**

The font size used for bold text.

---

int **bold_italics_font_size**

The font size used for bold italics text.

---

int **italics_font_size** 🔗

The font size used for italics text.

int **mono_font_size** 🔗

The font size used for monospace text.

int **normal_font_size** 🔗

The default text font size.

StyleBox **focus** 🔗

The background used when the **RichTextLabel** is focused. The focus StyleBox is displayed *over* the base StyleBox, so a partially transparent StyleBox should be used to ensure the base StyleBox remains visible. A StyleBox that represents an outline or an underline works well for this purpose. To disable the focus visual effect, assign a StyleBoxEmpty resource. Note that disabling the focus visual effect will harm keyboard/controller navigation usability, so this is not recommended for accessibility reasons.

StyleBox **normal** 🔗

The normal background for the **RichTextLabel**.

RigidBody2D

Inherits: PhysicsBody2D **<** CollisionObject2D **<** Node2D **<** CanvasItem **<** Node **<** Object

Inherited By: PhysicalBone2D

A 2D physics body that is moved by a physics simulation.

Description

RigidBody2D implements full 2D physics. It cannot be controlled directly, instead, you must apply forces to it (gravity, impulses, etc.), and the physics simulation will calculate the resulting movement, rotation, react to collisions, and affect other physics bodies in its path.

The body's behavior can be adjusted via lock_rotation, freeze, and freeze_mode. By changing various properties of the object, such as mass, you can control how the physics simulation acts on it.

A rigid body will always maintain its shape and size, even when forces are applied to it. It is useful for objects that can be interacted with in an environment, such as a tree that can be knocked over or a stack of crates that can be pushed around.

If you need to override the default physics behavior, you can write a custom force integration function. See custom_integrator.

Note: Changing the 2D transform or linear_velocity of a **RigidBody2D** very often may lead to some unpredictable behaviors. If you need to directly affect the body, prefer _integrate_forces as it allows you to directly access the physics state.

Tutorials

- 2D Physics Platformer Demo [https://godotengine.org/asset-library/asset/2725]
- Instancing Demo [https://godotengine.org/asset-library/asset/2716]

Properties

float	angular_damp	`0.0`
DampMode	angular_damp_mode	`0`
float	angular_velocity	`0.0`
bool	can_sleep	`true`
Vector2	center_of_mass	`Vector2(0, 0)`
CenterOfMassMode	center_of_mass_mode	`0`
Vector2	constant_force	`Vector2(0, 0)`
float	constant_torque	`0.0`

bool	contact_monitor	`false`
CCDMode	continuous_cd	`0`
bool	custom_integrator	`false`
bool	freeze	`false`
FreezeMode	freeze_mode	`0`
float	gravity_scale	`1.0`
float	inertia	`0.0`
float	linear_damp	`0.0`
DampMode	linear_damp_mode	`0`
Vector2	linear_velocity	`Vector2(0, 0)`
bool	lock_rotation	`false`
float	mass	`1.0`
int	max_contacts_reported	`0`

PhysicsMaterial	physics_material_override	
bool	sleeping	`false`

Methods

void	_integrate_forces(state: PhysicsDirectBodyState2D) virtual
void	add_constant_central_force(force: Vector2)
void	add_constant_force(force: Vector2, position: Vector2 = Vector2(0, 0))
void	add_constant_torque(torque: float)
void	apply_central_force(force: Vector2)
void	apply_central_impulse(impulse: Vector2 = Vector2(0, 0))
void	apply_force(force: Vector2, position: Vector2 = Vector2(0, 0))

void	apply_impulse(impulse: Vector2, position: Vector2 = Vector2(0, 0))
void	apply_torque(torque: float)
void	apply_torque_impulse(torque: float)
Array[Node2D]	get_colliding_bodies() const
int	get_contact_count() const
void	set_axis_velocity(axis_velocity: Vector2)

Signals

body_entered(body: Node) 🔗

Emitted when a collision with another PhysicsBody2D or TileMap occurs. Requires contact_monitor to be set to `true` and max_contacts_reported to be set high enough to detect all the collisions. TileMaps are detected if the TileSet has Collision Shape2Ds.

`body` the Node, if it exists in the tree, of the other PhysicsBody2D or TileMap.

body_exited(body: Node) 🔗

Emitted when the collision with another PhysicsBody2D or TileMap ends. Requires contact_monitor to be set to `true` and max_contacts_reported to be set high enough to detect all the collisions. TileMaps are detected if the TileSet has Collision Shape2Ds.

`body` the Node, if it exists in the tree, of the other PhysicsBody2D or TileMap.

---

body_shape_entered(body_rid: RID, body: Node, body_shape_index: int, local_shape_index: int) 🔗

Emitted when one of this RigidBody2D's Shape2Ds collides with another PhysicsBody2D or TileMap's Shape2Ds. Requires contact_monitor to be set to `true` and max_contacts_reported to be set high enough to detect all the collisions. TileMaps are detected if the TileSet has Collision Shape2Ds.

`body_rid` the RID of the other PhysicsBody2D or TileSet's CollisionObject2D used by the PhysicsServer2D.

`body` the Node, if it exists in the tree, of the other PhysicsBody2D or TileMap.

`body_shape_index` the index of the Shape2D of the other PhysicsBody2D or TileMap used by the PhysicsServer2D. Get the CollisionShape2D node with `body.shape_owner_get_owner(body.shape_find_owner(body_shape_index))`.

`local_shape_index` the index of the Shape2D of this RigidBody2D used by the PhysicsServer2D. Get the CollisionShape2D node with `self.shape_owner_get_owner(self.shape_find_owner(local_shape_index))`.

---

body_shape_exited(body_rid: RID, body: Node, body_shape_index: int, local_shape_index: int) 🔗

Emitted when the collision between one of this RigidBody2D's Shape2Ds and another PhysicsBody2D or TileMap's Shape2Ds ends. Requires contact_monitor to be set to `true` and max_contacts_reported to be set high enough to detect all the collisions. TileMaps are detected if the TileSet has Collision Shape2Ds.

`body_rid` the RID of the other PhysicsBody2D or TileSet's CollisionObject2D used by the PhysicsServer2D.

`body` the Node, if it exists in the tree, of the other PhysicsBody2D or TileMap.

`body_shape_index` the index of the Shape2D of the other PhysicsBody2D or TileMap used by the PhysicsServer2D. Get the CollisionShape2D node with `body.shape_owner_get_owner(body.shape_find_owner(body_shape_index))`.

`local_shape_index` the index of the Shape2D of this RigidBody2D used by the PhysicsServer2D. Get the CollisionShape2D node with `self.shape_owner_get_owner(self.shape_find_owner(local_shape_index))`.

---

sleeping_state_changed() 🔗

Emitted when the physics engine changes the body's sleeping state.

Note: Changing the value sleeping will not trigger this signal. It is only emitted if the sleeping state is changed by the physics engine or `emit_signal("sleeping_state_changed")` is used.

Enumerations

enum **FreezeMode**: 🔗

FreezeMode **FREEZE_MODE_STATIC** = `0`

Static body freeze mode (default). The body is not affected by gravity and forces. It can be only moved by user code and doesn't collide with other bodies along its path.

FreezeMode **FREEZE_MODE_KINEMATIC** = `1`

Kinematic body freeze mode. Similar to FREEZE_MODE_STATIC, but collides with other bodies along its path when moved. Useful for a frozen body that needs to be animated.

enum **CenterOfMassMode**: 🔗

CenterOfMassMode **CENTER_OF_MASS_MODE_AUTO** = `0`

In this mode, the body's center of mass is calculated automatically based on its shapes. This assumes that the shapes' origins are also their center of mass.

CenterOfMassMode **CENTER_OF_MASS_MODE_CUSTOM** = `1`

In this mode, the body's center of mass is set through center_of_mass. Defaults to the body's origin position.

enum **DampMode**: 🔗

DampMode **DAMP_MODE_COMBINE** = `0`

In this mode, the body's damping value is added to any value set in areas or the default value.

DampMode **DAMP_MODE_REPLACE** = `1`

In this mode, the body's damping value replaces any value set in areas or the default value.

---

enum **CCDMode**:

CCDMode **CCD_MODE_DISABLED** = `0`

Continuous collision detection disabled. This is the fastest way to detect body collisions, but can miss small, fast-moving objects.

CCDMode **CCD_MODE_CAST_RAY** = `1`

Continuous collision detection enabled using raycasting. This is faster than shapecasting but less precise.

CCDMode **CCD_MODE_CAST_SHAPE** = `2`

Continuous collision detection enabled using shapecasting. This is the slowest CCD method and the most precise.

Property Descriptions

float **angular_damp** = `0.0`

- void **set_angular_damp**(value: float)
- float **get_angular_damp**()

Damps the body's rotation. By default, the body will use the **Default Angular Damp** in **Project > Project Settings > Physics > 2d** or any value override set by an Area2D the body is in. Depending on angular_damp_mode, you can set angular_damp to be added to or to replace the body's damping value.

See ProjectSettings.physics/2d/default_angular_damp for more details about damping.

DampMode **angular_damp_mode** = `0`

- void **set_angular_damp_mode**(value: DampMode)
- DampMode **get_angular_damp_mode**()

Defines how angular_damp is applied. See DampMode for possible values.

---

float **angular_velocity** = `0.0`

- void **set_angular_velocity**(value: float)
- float **get_angular_velocity**()

The body's rotational velocity in *radians* per second.

---

bool **can_sleep** = `true`

- void **set_can_sleep**(value: bool)
- bool **is_able_to_sleep**()

If `true`, the body can enter sleep mode when there is no movement. See sleeping.

Vector2 **center_of_mass** = `Vector2(0, 0)`

- void **set_center_of_mass**(value: Vector2)

- Vector2 **get_center_of_mass**()

The body's custom center of mass, relative to the body's origin position, when center_of_mass_mode is set to CENTER_OF_MASS_MODE_CUSTOM. This is the balanced point of the body, where applied forces only cause linear acceleration. Applying forces outside of the center of mass causes angular acceleration.

When center_of_mass_mode is set to CENTER_OF_MASS_MODE_AUTO (default value), the center of mass is automatically computed.

CenterOfMassMode **center_of_mass_mode** = `0` 🔗

- void **set_center_of_mass_mode**(value: CenterOfMassMode)
- CenterOfMassMode **get_center_of_mass_mode**()

Defines the way the body's center of mass is set. See CenterOfMassMode for possible values.

Vector2 **constant_force** = `Vector2(0, 0)` 🔗

- void **set_constant_force**(value: Vector2)
- Vector2 **get_constant_force**()

The body's total constant positional forces applied during each physics update.

See add_constant_force and add_constant_central_force.

---

float **constant_torque** = `0.0` 🔗

- void **set_constant_torque**(value: float)

- float **get_constant_torque**()

The body's total constant rotational forces applied during each physics update.

See add_constant_torque.

bool **contact_monitor** = `false` 🔗

- void **set_contact_monitor**(value: bool)
- bool **is_contact_monitor_enabled**()

If `true`, the RigidBody2D will emit signals when it collides with another body.

Note: By default the maximum contacts reported is set to 0, meaning nothing will be recorded, see max_contacts_reported.

CCDMode **continuous_cd** = `0` 🔗

- void **set_continuous_collision_detection_mode**(value: CCDMode)
- CCDMode **get_continuous_collision_detection_mode**()

Continuous collision detection mode.

Continuous collision detection tries to predict where a moving body will collide instead of moving it and correcting its movement after collision. Continuous collision detection is slower, but more precise and misses fewer collisions with small, fast-moving objects. Raycasting and shapecasting methods are available. See CCDMode for details.

bool **custom_integrator** = `false` 🔗

- void **set_use_custom_integrator**(value: bool)
- bool **is_using_custom_integrator**()

If `true`, the standard force integration (like gravity or damping) will be disabled for this body. Other than collision response, the body will only move as determined by the _integrate_forces method, if that virtual method is overridden.

Setting this property will call the method PhysicsServer2D.body_set_omit_force_integration internally.

---

bool **freeze** = `false`

- void **set_freeze_enabled**(value: bool)
- bool **is_freeze_enabled**()

If `true`, the body is frozen. Gravity and forces are not applied anymore.

See freeze_mode to set the body's behavior when frozen.

For a body that is always frozen, use StaticBody2D or AnimatableBody2D instead.

FreezeMode **freeze_mode** = `0`

- void **set_freeze_mode**(value: FreezeMode)
- FreezeMode **get_freeze_mode**()

The body's freeze mode. Can be used to set the body's behavior when freeze is enabled. See FreezeMode for possible values.

For a body that is always frozen, use StaticBody2D or AnimatableBody2D instead.

---

float **gravity_scale** = `1.0`

- void **set_gravity_scale**(value: float)
- float **get_gravity_scale**()

Multiplies the gravity applied to the body. The body's gravity is calculated from the **Default Gravity** value in **Project > Project Settings > Physics > 2d** and/or any additional gravity vector applied by Area2Ds.

---

float **inertia** = `0.0`

- void **set_inertia**(value: float)
- float **get_inertia**()

The body's moment of inertia. This is like mass, but for rotation: it determines how much torque it takes to rotate the body. The moment of inertia is usually computed automatically from the mass and the shapes, but this property allows you to set a custom value.

If set to `0`, inertia is automatically computed (default value).

Note: This value does not change when inertia is automatically computed. Use PhysicsServer2D to get the computed inertia.

GDScript

```
@onready var ball = $Ball

func get_ball_inertia():
    return 1.0 / PhysicsServer2D.body_get_direct_state(ball.get_rid()).inverse_inertia
```

C#

```csharp
private RigidBody2D _ball;

public override void _Ready()
{
    _ball = GetNode<RigidBody2D>("Ball");
}

private float GetBallInertia()
{
    return 1.0f / PhysicsServer2D.BodyGetDirectState(_ball.GetRid()).InverseInertia;
}
```

float **linear_damp** = `0.0`

- void **set_linear_damp**(value: float)
- float **get_linear_damp**()

Damps the body's movement. By default, the body will use the **Default Linear Damp** in **Project > Project Settings > Physics > 2d** or any value override set by an Area2D the body is in. Depending on linear_damp_mode, you can set linear_damp to be added to or to replace the body's damping value.

See ProjectSettings.physics/2d/default_linear_damp for more details about damping.

DampMode **linear_damp_mode** = `0`

- void **set_linear_damp_mode**(value: DampMode)
- DampMode **get_linear_damp_mode**()

Defines how linear_damp is applied. See DampMode for possible values.

Vector2 **linear_velocity** = `Vector2(0, 0)`

- void **set_linear_velocity**(value: Vector2)
- Vector2 **get_linear_velocity**()

The body's linear velocity in pixels per second. Can be used sporadically, but **don't set this every frame**, because physics may run in another thread and runs at a different granularity. Use _integrate_forces as your process loop for precise control of the body state.

bool **lock_rotation** = `false`

- void **set_lock_rotation_enabled**(value: bool)
- bool **is_lock_rotation_enabled**()

If `true`, the body cannot rotate. Gravity and forces only apply linear movement.

float **mass** = `1.0`

- void **set_mass**(value: float)
- float **get_mass**()

The body's mass.

int **max_contacts_reported** = `0`

- void **set_max_contacts_reported**(value: int)
- int **get_max_contacts_reported**()

The maximum number of contacts that will be recorded. Requires a value greater than 0 and contact_monitor to be set to `true` to start to register contacts. Use get_contact_count to retrieve the count or get_colliding_bodies to retrieve bodies that have been collided with.

Note: The number of contacts is different from the number of collisions. Collisions between parallel edges will result in two contacts (one at each end), and collisions between parallel faces will result in four contacts (one at each corner).

PhysicsMaterial **physics_material_override**

- void **set_physics_material_override**(value: PhysicsMaterial)
- PhysicsMaterial **get_physics_material_override**()

The physics material override for the body.

If a material is assigned to this property, it will be used instead of any other physics material, such as an inherited one.

---

bool **sleeping** = `false`

- void **set_sleeping**(value: bool)
- bool **is_sleeping**()

If `true`, the body will not move and will not calculate forces until woken up by another body through, for example, a collision, or by using the apply_impulse or apply_force methods.

Method Descriptions

void **_integrate_forces**(state: PhysicsDirectBodyState2D) virtual

Called during physics processing, allowing you to read and safely modify the simulation state for the object. By default, it is called before the standard force integration, but the custom_integrator property allows you to disable the standard force integration and do fully custom force integration for a body.

---

void **add_constant_central_force**(force: Vector2) 🔗

Adds a constant directional force without affecting rotation that keeps being applied over time until cleared with `constant_force = Vector2(0, 0)`.

This is equivalent to using add_constant_force at the body's center of mass.

---

void **add_constant_force**(force: Vector2, position: Vector2 = Vector2(0, 0)) 🔗

Adds a constant positioned force to the body that keeps being applied over time until cleared with `constant_force = Vector2(0, 0)`.

`position` is the offset from the body origin in global coordinates.

---

void **add_constant_torque**(torque: float) 🔗

Adds a constant rotational force without affecting position that keeps being applied over time until cleared with `constant_torque = 0`.

---

void **apply_central_force**(force: Vector2) 🔗

Applies a directional force without affecting rotation. A force is time dependent and meant to be applied every physics update.

This is equivalent to using apply_force at the body's center of mass.

---

void **apply_central_impulse**(impulse: Vector2 = Vector2(0, 0))

Applies a directional impulse without affecting rotation.

An impulse is time-independent! Applying an impulse every frame would result in a framerate-dependent force. For this reason, it should only be used when simulating one-time impacts (use the "_force" functions otherwise).

This is equivalent to using apply_impulse at the body's center of mass.

---

void **apply_force**(force: Vector2, position: Vector2 = Vector2(0, 0))

Applies a positioned force to the body. A force is time dependent and meant to be applied every physics update.

`position` is the offset from the body origin in global coordinates.

---

void **apply_impulse**(impulse: Vector2, position: Vector2 = Vector2(0, 0))

Applies a positioned impulse to the body.

An impulse is time-independent! Applying an impulse every frame would result in a framerate-dependent force. For this reason, it should only be used when simulating one-time impacts (use the "_force" functions otherwise).

`position` is the offset from the body origin in global coordinates.

---

void **apply_torque**(torque: float) 🔗

Applies a rotational force without affecting position. A force is time dependent and meant to be applied every physics update.

Note: inertia is required for this to work. To have inertia, an active CollisionShape2D must be a child of the node, or you can manually set inertia.

---

void **apply_torque_impulse**(torque: float) 🔗

Applies a rotational impulse to the body without affecting the position.

An impulse is time-independent! Applying an impulse every frame would result in a framerate-dependent force. For this reason, it should only be used when simulating one-time impacts (use the "_force" functions otherwise).

Note: inertia is required for this to work. To have inertia, an active CollisionShape2D must be a child of the node, or you can manually set inertia.

Array[Node2D] **get_colliding_bodies**() const 🔗

Returns a list of the bodies colliding with this one. Requires contact_monitor to be set to `true` and max_contacts_reported to be set high enough to detect all the collisions.

Note: The result of this test is not immediate after moving objects. For performance, list of collisions is updated once per frame and before the physics step. Consider using signals instead.

---

int **get_contact_count**() const

Returns the number of contacts this body has with other bodies. By default, this returns 0 unless bodies are configured to monitor contacts (see contact_monitor).

Note: To retrieve the colliding bodies, use get_colliding_bodies.

---

void **set_axis_velocity**(axis_velocity: Vector2)

Sets the body's velocity on the given axis. The velocity in the given vector axis will be set as the given vector length. This is useful for jumping behavior.

RigidBody3D

Inherits: PhysicsBody3D **<** CollisionObject3D **<** Node3D **<** Node **<** Object

Inherited By: VehicleBody3D

A 3D physics body that is moved by a physics simulation.

Description

RigidBody3D implements full 3D physics. It cannot be controlled directly, instead, you must apply forces to it (gravity, impulses, etc.), and the physics simulation will calculate the resulting movement, rotation, react to collisions, and affect other physics bodies in its path.

The body's behavior can be adjusted via lock_rotation, freeze, and freeze_mode. By changing various properties of the object, such as mass, you can control how the physics simulation acts on it.

A rigid body will always maintain its shape and size, even when forces are applied to it. It is useful for objects that can be interacted with in an environment, such as a tree that can be knocked over or a stack of crates that can be pushed around.

If you need to override the default physics behavior, you can write a custom force integration function. See custom_integrator.

Note: Changing the 3D transform or linear_velocity of a **RigidBody3D** very often may lead to some unpredictable behaviors. If you need to directly affect the body, prefer _integrate_forces as it allows you to directly access the physics state.

Tutorials

- Physics introduction
- 3D Truck Town Demo [https://godotengine.org/asset-library/asset/2752]
- 3D Physics Tests Demo [https://godotengine.org/asset-library/asset/2747]

Properties

float	angular_damp	`0.0`
DampMode	angular_damp_mode	`0`
Vector3	angular_velocity	`Vector3(0, 0, 0)`
bool	can_sleep	`true`
Vector3	center_of_mass	`Vector3(0, 0, 0)`
CenterOfMassMode	center_of_mass_mode	`0`
Vector3	constant_force	`Vector3(0, 0, 0)`
Vector3	constant_torque	`Vector3(0, 0, 0)`

bool	contact_monitor	`false`
bool	continuous_cd	`false`
bool	custom_integrator	`false`
bool	freeze	`false`
FreezeMode	freeze_mode	`0`
float	gravity_scale	`1.0`
Vector3	inertia	`Vector3(0, 0, 0)`
float	linear_damp	`0.0`
DampMode	linear_damp_mode	`0`
Vector3	linear_velocity	`Vector3(0, 0, 0)`
bool	lock_rotation	`false`
float	mass	`1.0`

int	max_contacts_reported	`0`
PhysicsMaterial	physics_material_override	
bool	sleeping	`false`

Methods

void	_integrate_forces(state: PhysicsDirectBodyState3D) virtual
void	add_constant_central_force(force: Vector3)
void	add_constant_force(force: Vector3, position: Vector3 = Vector3(0, 0, 0))
void	add_constant_torque(torque: Vector3)
void	apply_central_force(force: Vector3)
void	apply_central_impulse(impulse: Vector3)
void	apply_force(force: Vector3, position: Vector3 = Vector3(0, 0, 0))

void	apply_impulse(impulse: Vector3, position: Vector3 = Vector3(0, 0, 0))
void	apply_torque(torque: Vector3)
void	apply_torque_impulse(impulse: Vector3)
Array[Node3D]	get_colliding_bodies() const
int	get_contact_count() const
Basis	get_inverse_inertia_tensor() const
void	set_axis_velocity(axis_velocity: Vector3)

Signals

body_entered(body: Node)

Emitted when a collision with another PhysicsBody3D or GridMap occurs. Requires contact_monitor to be set to `true` and max_contacts_reported to be set high enough to detect all the collisions. GridMaps are detected if the MeshLibrary has Collision Shape3Ds.

`body` the Node, if it exists in the tree, of the other PhysicsBody3D or GridMap.

---

body_exited(body: Node) 🔗

Emitted when the collision with another PhysicsBody3D or GridMap ends. Requires contact_monitor to be set to `true` and max_contacts_reported to be set high enough to detect all the collisions. GridMaps are detected if the MeshLibrary has Collision Shape3Ds.

`body` the Node, if it exists in the tree, of the other PhysicsBody3D or GridMap.

---

body_shape_entered(body_rid: RID, body: Node, body_shape_index: int, local_shape_index: int) 🔗

Emitted when one of this RigidBody3D's Shape3Ds collides with another PhysicsBody3D or GridMap's Shape3Ds. Requires contact_monitor to be set to `true` and max_contacts_reported to be set high enough to detect all the collisions. GridMaps are detected if the MeshLibrary has Collision Shape3Ds.

`body_rid` the RID of the other PhysicsBody3D or MeshLibrary's CollisionObject3D used by the PhysicsServer3D.

`body` the Node, if it exists in the tree, of the other PhysicsBody3D or GridMap.

`body_shape_index` the index of the Shape3D of the other PhysicsBody3D or GridMap used by the PhysicsServer3D. Get the CollisionShape3D node with `body.shape_owner_get_owner(body.shape_find_owner(body_shape_index))`.

`local_shape_index` the index of the Shape3D of this RigidBody3D used by the PhysicsServer3D. Get the CollisionShape3D node with `self.shape_owner_get_owner(self.shape_find_owner(local_shape_index))`.

---

body_shape_exited(body_rid: RID, body: Node, body_shape_index: int, local_shape_index: int) 🔗

Emitted when the collision between one of this RigidBody3D's Shape3Ds and another PhysicsBody3D or GridMap's Shape3Ds ends. Requires contact_monitor to be set to `true` and max_contacts_reported to be set high enough to detect all the collisions. GridMaps are detected if the MeshLibrary has Collision Shape3Ds.

`body_rid` the RID of the other PhysicsBody3D or MeshLibrary's CollisionObject3D used by the PhysicsServer3D. GridMaps are detected if the Meshes have Shape3Ds.

`body` the Node, if it exists in the tree, of the other PhysicsBody3D or GridMap.

`body_shape_index` the index of the Shape3D of the other PhysicsBody3D or GridMap used by the PhysicsServer3D. Get the CollisionShape3D node with `body.shape_owner_get_owner(body.shape_find_owner(body_shape_index))`.

`local_shape_index` the index of the Shape3D of this RigidBody3D used by the PhysicsServer3D. Get the CollisionShape3D node with `self.shape_owner_get_owner(self.shape_find_owner(local_shape_index))`.

---

sleeping_state_changed() 🔗

Emitted when the physics engine changes the body's sleeping state.

Note: Changing the value sleeping will not trigger this signal. It is only emitted if the sleeping state is changed by the physics engine or `emit_signal("sleeping_state_changed")` is used.

Enumerations

enum **FreezeMode**: 🔗

FreezeMode **FREEZE_MODE_STATIC** = `0`

Static body freeze mode (default). The body is not affected by gravity and forces. It can be only moved by user code and doesn't collide with other bodies along its path.

FreezeMode **FREEZE_MODE_KINEMATIC** = `1`

Kinematic body freeze mode. Similar to FREEZE_MODE_STATIC, but collides with other bodies along its path when moved. Useful for a frozen body that needs to be animated.

enum **CenterOfMassMode**: 🔗

CenterOfMassMode **CENTER_OF_MASS_MODE_AUTO** = `0`

In this mode, the body's center of mass is calculated automatically based on its shapes. This assumes that the shapes' origins are also their center of mass.

CenterOfMassMode **CENTER_OF_MASS_MODE_CUSTOM** = `1`

In this mode, the body's center of mass is set through center_of_mass. Defaults to the body's origin position.

enum **DampMode**:

DampMode **DAMP_MODE_COMBINE** = `0`

In this mode, the body's damping value is added to any value set in areas or the default value.

DampMode **DAMP_MODE_REPLACE** = `1`

In this mode, the body's damping value replaces any value set in areas or the default value.

Property Descriptions

float **angular_damp** = `0.0`

- void **set_angular_damp**(value: float)
- float **get_angular_damp**()

Damps the body's rotation. By default, the body will use the **Default Angular Damp** in **Project > Project Settings > Physics > 3d** or any value override set by an Area3D the body is in. Depending on angular_damp_mode, you can set angular_damp to be added to or to replace the body's damping value.

See ProjectSettings.physics/3d/default_angular_damp for more details about damping.

DampMode **angular_damp_mode** = `0`

- void **set_angular_damp_mode**(value: DampMode)
- DampMode **get_angular_damp_mode**()

Defines how angular_damp is applied. See DampMode for possible values.

Vector3 **angular_velocity** = `Vector3(0, 0, 0)` 🔗

- void **set_angular_velocity**(value: Vector3)
- Vector3 **get_angular_velocity**()

The RigidBody3D's rotational velocity in *radians* per second.

---

bool **can_sleep** = `true` 🔗

- void **set_can_sleep**(value: bool)
- bool **is_able_to_sleep**()

If `true`, the body can enter sleep mode when there is no movement. See sleeping.

Vector3 **center_of_mass** = `Vector3(0, 0, 0)` 🔗

- void **set_center_of_mass**(value: Vector3)
- Vector3 **get_center_of_mass**()

The body's custom center of mass, relative to the body's origin position, when center_of_mass_mode is set to CENTER_OF_MASS_MODE_CUSTOM. This is the balanced point of the body, where applied forces only cause linear acceleration. Applying forces outside of the center of mass causes angular acceleration.

When center_of_mass_mode is set to CENTER_OF_MASS_MODE_AUTO (default value), the center of mass is automatically computed.

CenterOfMassMode **center_of_mass_mode** = `0`

- void **set_center_of_mass_mode**(value: CenterOfMassMode)
- CenterOfMassMode **get_center_of_mass_mode**()

Defines the way the body's center of mass is set. See CenterOfMassMode for possible values.

Vector3 **constant_force** = `Vector3(0, 0, 0)`

- void **set_constant_force**(value: Vector3)
- Vector3 **get_constant_force**()

The body's total constant positional forces applied during each physics update.

See add_constant_force and add_constant_central_force.

Vector3 **constant_torque** = `Vector3(0, 0, 0)`

- void **set_constant_torque**(value: Vector3)
- Vector3 **get_constant_torque**()

The body's total constant rotational forces applied during each physics update.

See add_constant_torque.

---

bool **contact_monitor** = `false`

- void **set_contact_monitor**(value: bool)
- bool **is_contact_monitor_enabled**()

If `true`, the RigidBody3D will emit signals when it collides with another body.

Note: By default the maximum contacts reported is set to 0, meaning nothing will be recorded, see max_contacts_reported.

bool **continuous_cd** = `false`

- void **set_use_continuous_collision_detection**(value: bool)
- bool **is_using_continuous_collision_detection**()

If `true`, continuous collision detection is used.

Continuous collision detection tries to predict where a moving body will collide, instead of moving it and correcting its movement if it collided. Continuous collision detection is more precise, and misses fewer impacts by small, fast-moving objects. Not using continuous collision detection is faster to compute, but can miss small, fast-moving objects.

bool **custom_integrator** = `false`

- void **set_use_custom_integrator**(value: bool)
- bool **is_using_custom_integrator**()

If `true`, the standard force integration (like gravity or damping) will be disabled for this body. Other than collision response, the body will only move as determined by the _integrate_forces method, if that virtual method is overridden.

Setting this property will call the method PhysicsServer3D.body_set_omit_force_integration internally.

bool **freeze** = `false`

- void **set_freeze_enabled**(value: bool)
- bool **is_freeze_enabled**()

If `true`, the body is frozen. Gravity and forces are not applied anymore.

See freeze_mode to set the body's behavior when frozen.

For a body that is always frozen, use StaticBody3D or AnimatableBody3D instead.

FreezeMode **freeze_mode** = `0` 🔗

- void **set_freeze_mode**(value: FreezeMode)
- FreezeMode **get_freeze_mode**()

The body's freeze mode. Can be used to set the body's behavior when freeze is enabled. See FreezeMode for possible values.

For a body that is always frozen, use StaticBody3D or AnimatableBody3D instead.

float **gravity_scale** = `1.0` 🔗

- void **set_gravity_scale**(value: float)
- float **get_gravity_scale**()

This is multiplied by the global 3D gravity setting found in **Project > Project Settings > Physics > 3d** to produce RigidBody3D's gravity. For example, a value of 1 will be normal gravity, 2 will apply double gravity, and 0.5 will apply half gravity to this object.

Vector3 **inertia** = `Vector3(0, 0, 0)` 🔗

- void **set_inertia**(value: Vector3)
- Vector3 **get_inertia**()

The body's moment of inertia. This is like mass, but for rotation: it determines how much torque it takes to rotate the body on each axis. The moment of inertia is usually computed automatically from the mass and the shapes, but this property allows you to set a custom value.

If set to Vector3.ZERO, inertia is automatically computed (default value).

Note: This value does not change when inertia is automatically computed. Use PhysicsServer3D to get the computed inertia.

GDScript

```
@onready var ball = $Ball

func get_ball_inertia():
    return PhysicsServer3D.body_get_direct_state(ball.get_rid()).inverse_inertia.inverse()
```

C#

```
private RigidBody3D _ball;

public override void _Ready()
{
    _ball = GetNode<RigidBody3D>("Ball");
}

private Vector3 GetBallInertia()
{
    return PhysicsServer3D.BodyGetDirectState(_ball.GetRid()).InverseInertia.Inverse();
}
```

---

float **linear_damp** = `0.0` 🔗

- void **set_linear_damp**(value: float)
- float **get_linear_damp**()

Damps the body's movement. By default, the body will use the **Default Linear Damp** in **Project > Project Settings > Physics > 3d** or any value override set by an Area3D the body is in. Depending on linear_damp_mode, you can set linear_damp to be added to or to replace the body's damping value.

See ProjectSettings.physics/3d/default_linear_damp for more details about damping.

DampMode **linear_damp_mode** = `0` 🔗

- void **set_linear_damp_mode**(value: DampMode)
- DampMode **get_linear_damp_mode**()

Defines how linear_damp is applied. See DampMode for possible values.

Vector3 **linear_velocity** = `Vector3(0, 0, 0)` 🔗

- void **set_linear_velocity**(value: Vector3)
- Vector3 **get_linear_velocity**()

The body's linear velocity in units per second. Can be used sporadically, but **don't set this every frame**, because physics may run in another thread and runs at a different granularity. Use _integrate_forces as your process loop for precise control of the body state.

---

bool **lock_rotation** = `false` 🔗

- void **set_lock_rotation_enabled**(value: bool)
- bool **is_lock_rotation_enabled**()

If `true`, the body cannot rotate. Gravity and forces only apply linear movement.

---

float **mass** = `1.0`

- void **set_mass**(value: float)
- float **get_mass**()

The body's mass.

---

int **max_contacts_reported** = `0`

- void **set_max_contacts_reported**(value: int)
- int **get_max_contacts_reported**()

The maximum number of contacts that will be recorded. Requires a value greater than 0 and contact_monitor to be set to `true` to start to register contacts. Use get_contact_count to retrieve the count or get_colliding_bodies to retrieve bodies that have been collided with.

Note: The number of contacts is different from the number of collisions. Collisions between parallel edges will result in two contacts (one at each end), and collisions between parallel faces will result in four contacts (one at each corner).

PhysicsMaterial **physics_material_override**

- void **set_physics_material_override**(value: PhysicsMaterial)
- PhysicsMaterial **get_physics_material_override**()

The physics material override for the body.

If a material is assigned to this property, it will be used instead of any other physics material, such as an inherited one.

---

bool **sleeping** = `false`

- void **set_sleeping**(value: bool)
- bool **is_sleeping**()

If `true`, the body will not move and will not calculate forces until woken up by another body through, for example, a collision, or by using the apply_impulse or apply_force methods.

Method Descriptions

void **_integrate_forces**(state: PhysicsDirectBodyState3D) virtual

Called during physics processing, allowing you to read and safely modify the simulation state for the object. By default, it is called before the standard force integration, but the custom_integrator property allows you to disable the standard force integration and do fully custom force integration for a body.

---

void **add_constant_central_force**(force: Vector3)

Adds a constant directional force without affecting rotation that keeps being applied over time until cleared with `constant_force = Vector3(0, 0, 0)`.

This is equivalent to using add_constant_force at the body's center of mass.

---

void **add_constant_force**(force: Vector3, position: Vector3 = Vector3(0, 0, 0)) 🔗

Adds a constant positioned force to the body that keeps being applied over time until cleared with `constant_force = Vector3(0, 0, 0)`.

`position` is the offset from the body origin in global coordinates.

---

void **add_constant_torque**(torque: Vector3) 🔗

Adds a constant rotational force without affecting position that keeps being applied over time until cleared with `constant_torque = Vector3(0, 0, 0)`.

---

void **apply_central_force**(force: Vector3) 🔗

Applies a directional force without affecting rotation. A force is time dependent and meant to be applied every physics update.

This is equivalent to using apply_force at the body's center of mass.

---

void **apply_central_impulse**(impulse: Vector3) 🔗

Applies a directional impulse without affecting rotation.

An impulse is time-independent! Applying an impulse every frame would result in a framerate-dependent force. For this reason, it should only be used when simulating one-time impacts (use the "_force" functions otherwise).

This is equivalent to using apply_impulse at the body's center of mass.

---

void **apply_force**(force: Vector3, position: Vector3 = Vector3(0, 0, 0)) 🔗

Applies a positioned force to the body. A force is time dependent and meant to be applied every physics update.

`position` is the offset from the body origin in global coordinates.

void **apply_impulse**(impulse: Vector3, position: Vector3 = Vector3(0, 0, 0)) 🔗

Applies a positioned impulse to the body.

An impulse is time-independent! Applying an impulse every frame would result in a framerate-dependent force. For this reason, it should only be used when simulating one-time impacts (use the "_force" functions otherwise).

`position` is the offset from the body origin in global coordinates.

void **apply_torque**(torque: Vector3) 🔗

Applies a rotational force without affecting position. A force is time dependent and meant to be applied every physics update.

Note: inertia is required for this to work. To have inertia, an active CollisionShape3D must be a child of the node, or you can manually set inertia.

void **apply_torque_impulse**(impulse: Vector3) 🔗

Applies a rotational impulse to the body without affecting the position.

An impulse is time-independent! Applying an impulse every frame would result in a framerate-dependent force. For this reason, it should only be used when simulating one-time impacts (use the "_force" functions otherwise).

Note: inertia is required for this to work. To have inertia, an active CollisionShape3D must be a child of the node, or you can manually set inertia.

Array[Node3D] **get_colliding_bodies**() const 🔗

Returns a list of the bodies colliding with this one. Requires contact_monitor to be set to `true` and max_contacts_reported to be set high enough to detect all the collisions.

Note: The result of this test is not immediate after moving objects. For performance, list of collisions is updated once per frame and before the physics step. Consider using signals instead.

int **get_contact_count**() const 🔗

Returns the number of contacts this body has with other bodies. By default, this returns 0 unless bodies are configured to monitor contacts (see contact_monitor).

Note: To retrieve the colliding bodies, use get_colliding_bodies.

Basis **get_inverse_inertia_tensor**() const 🔗

Returns the inverse inertia tensor basis. This is used to calculate the angular acceleration resulting from a torque applied to the **RigidBody3D**.

void **set_axis_velocity**(axis_velocity: Vector3)

Sets an axis velocity. The velocity in the given vector axis will be set as the given vector length. This is useful for jumping behavior.

RootMotionView

Inherits: VisualInstance3D **<** Node3D **<** Node **<** Object

Editor-only helper for setting up root motion in AnimationMixer.

Description

Root motion refers to an animation technique where a mesh's skeleton is used to give impulse to a character. When working with 3D animations, a popular technique is for animators to use the root skeleton bone to give motion to the rest of the skeleton. This allows animating characters in a way where steps actually match the floor below. It also allows precise interaction with objects during cinematics. See also AnimationMixer.

Note: RootMotionView is only visible in the editor. It will be hidden automatically in the running project.

Tutorials

- Using AnimationTree - Root motion

Properties

NodePath	animation_path	`NodePath("")`
float	cell_size	`1.0`

Color	color	`Color(0.5, 0.5, 1, 1)`
float	radius	`10.0`
bool	zero_y	`true`

Property Descriptions

NodePath **animation_path** = `NodePath("")` 🔗

- void **set_animation_path**(value: NodePath)
- NodePath **get_animation_path**()

Path to an AnimationMixer node to use as a basis for root motion.

---

float **cell_size** = `1.0` 🔗

- void **set_cell_size**(value: float)
- float **get_cell_size**()

The grid's cell size in 3D units.

Color **color** = `Color(0.5, 0.5, 1, 1)` 🔗

- void **set_color**(value: Color)
- Color **get_color**()

The grid's color.

---

float **radius** = `10.0`

- void **set_radius**(value: float)
- float **get_radius**()

The grid's radius in 3D units. The grid's opacity will fade gradually as the distance from the origin increases until this radius is reached.

---

bool **zero_y** = `true`

- void **set_zero_y**(value: bool)
- bool **get_zero_y**()

If `true`, the grid's points will all be on the same Y coordinate (*local* Y = 0). If `false`, the points' original Y coordinate is preserved.

ScriptCreateDialog

Inherits: ConfirmationDialog **<** AcceptDialog **<** Window **<** Viewport **<** Node **<** Object

Godot editor's popup dialog for creating new Script files.

Description

The **ScriptCreateDialog** creates script files according to a given template for a given scripting language. The standard use is to configure its fields prior to calling one of the Window.popup methods.

GDScript

```
func _ready():
    var dialog = ScriptCreateDialog.new();
    dialog.config("Node", "res://new_node.gd") # For in-engine types.
    dialog.config("\"res://base_node.gd\"", "res://derived_node.gd") # For script types.
    dialog.popup_centered()
```

C#

```
public override void _Ready()
{
    var dialog = new ScriptCreateDialog();
    dialog.Config("Node", "res://NewNode.cs"); // For in-engine types.
    dialog.Config("\"res://BaseNode.cs\"", "res://DerivedNode.cs"); // For script types.
    dialog.PopupCentered();
}
```

Properties

bool	dialog_hide_on_ok	`false` (overrides AcceptDialog)
String	ok_button_text	`"Create"` (overrides AcceptDialog)
String	title	`"Attach Node Script"` (overrides Window)

Methods

void	config(inherits: String, path: String, built_in_enabled: bool = true, load_enabled: bool = true)

Signals

script_created(script: Script)

Emitted when the user clicks the OK button.

Method Descriptions

void **config**(inherits: String, path: String, built_in_enabled: bool = true, load_enabled: bool = true)

Prefills required fields to configure the ScriptCreateDialog for use.

ScriptEditor

Inherits: PanelContainer **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

Godot editor's script editor.

Description

Godot editor's script editor.

Note: This class shouldn't be instantiated directly. Instead, access the singleton using EditorInterface.get_script_editor.

Methods

ScriptEditorBase	get_current_editor() const
Script	get_current_script()
Array[ScriptEditorBase]	get_open_script_editors() const
Array[Script]	get_open_scripts() const
void	goto_help(topic: String)
void	goto_line(line_number: int)

void	open_script_create_dialog(base_name: String, base_path: String)
void	register_syntax_highlighter(syntax_highlighter: EditorSyntaxHighlighter)
void	unregister_syntax_highlighter(syntax_highlighter: EditorSyntaxHighlighter)

Signals

editor_script_changed(script: Script)

Emitted when user changed active script. Argument is a freshly activated Script.

script_close(script: Script)

Emitted when editor is about to close the active script. Argument is a Script that is going to be closed.

Method Descriptions

ScriptEditorBase **get_current_editor**() const

Returns the ScriptEditorBase object that the user is currently editing.

Script **get_current_script**()

Returns a Script that is currently active in editor.

Array[ScriptEditorBase] **get_open_script_editors**() const

Returns an array with all ScriptEditorBase objects which are currently open in editor.

Array[Script] **get_open_scripts**() const

Returns an array with all Script objects which are currently open in editor.

---

void **goto_help**(topic: String)

Opens help for the given topic. The `topic` is an encoded string that controls which class, method, constant, signal, annotation, property, or theme item should be focused.

The supported `topic` formats include `class_name:class`, `class_method:class:method`, `class_constant:class:constant`, `class_signal:class:signal`, `class_annotation:class:@annotation`, `class_property:class:property`, and `class_theme_item:class:item`, where `class` is the class name, `method` is the method name, `constant` is the constant name, `signal` is the signal name, `annotation` is the annotation name, `property` is the property name, and `item` is the theme item.

Examples:

```
# Shows help for the Node class.
class_name:Node
# Shows help for the global min function.
# Global objects are accessible in the `@GlobalScope` namespace, shown here.
class_method:@GlobalScope:min
# Shows help for get_viewport in the Node class.
class_method:Node:get_viewport
# Shows help for the Input constant MOUSE_BUTTON_MIDDLE.
class_constant:Input:MOUSE_BUTTON_MIDDLE
# Shows help for the BaseButton signal pressed.
class_signal:BaseButton:pressed
# Shows help for the CanvasItem property visible.
```

```
class_property:CanvasItem:visible
# Shows help for the GDScript annotation export.
# Annotations should be prefixed with the `@` symbol in the descriptor, as shown here.
class_annotation:@GDScript:@export
# Shows help for the GraphNode theme item named panel_selected.
class_theme_item:GraphNode:panel_selected
```

---

void **goto_line**(line_number: int)

Goes to the specified line in the current script.

---

void **open_script_create_dialog**(base_name: String, base_path: String)

Opens the script create dialog. The script will extend `base_name`. The file extension can be omitted from `base_path`. It will be added based on the selected scripting language.

---

void **register_syntax_highlighter**(syntax_highlighter: EditorSyntaxHighlighter)

Registers the EditorSyntaxHighlighter to the editor, the EditorSyntaxHighlighter will be available on all open scripts.

Note: Does not apply to scripts that are already opened.

---

void **unregister_syntax_highlighter**(syntax_highlighter: EditorSyntaxHighlighter)

Unregisters the EditorSyntaxHighlighter from the editor.

Note: The EditorSyntaxHighlighter will still be applied to scripts that are already opened.

ScriptEditorBase

Inherits: VBoxContainer **<** BoxContainer **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

Base editor for editing scripts in the ScriptEditor.

Description

Base editor for editing scripts in the ScriptEditor. This does not include documentation items.

Methods

void	add_syntax_highlighter(highlighter: EditorSyntaxHighlighter)
Control	get_base_editor() const

Signals

edited_script_changed() 🔗

Emitted after script validation.

go_to_help(what: String) 🔗

Emitted when the user requests a specific documentation page.

go_to_method(script: Object, method: String)

Emitted when the user requests to view a specific method of a script, similar to request_open_script_at_line.

name_changed()

Emitted after script validation or when the edited resource has changed.

replace_in_files_requested(text: String)

Emitted when the user request to find and replace text in the file system.

request_help(topic: String)

Emitted when the user requests contextual help.

request_open_script_at_line(script: Object, line: int)

Emitted when the user requests to view a specific line of a script, similar to go_to_method.

request_save_history()

Emitted when the user contextual goto and the item is in the same script.

request_save_previous_state(state: Dictionary)

Emitted when the user changes current script or moves caret by 10 or more columns within the same script.

---

search_in_files_requested(text: String)

Emitted when the user request to search text in the file system.

Method Descriptions

void **add_syntax_highlighter**(highlighter: EditorSyntaxHighlighter)

Adds a EditorSyntaxHighlighter to the open script.

Control **get_base_editor**() const

Returns the underlying Control used for editing scripts. For text scripts, this is a CodeEdit.

ScrollBar

Inherits: Range **<** Control **<** CanvasItem **<** Node **<** Object

Inherited By: HScrollBar, VScrollBar

Abstract base class for scrollbars.

Description

Abstract base class for scrollbars, typically used to navigate through content that extends beyond the visible area of a control. Scrollbars are Range-based controls.

Properties

float	custom_step	`-1.0`
float	step	`0.0` (overrides Range)

Theme Properties

Texture2D	decrement
Texture2D	decrement_highlight

Texture2D	decrement_pressed
Texture2D	increment
Texture2D	increment_highlight
Texture2D	increment_pressed
StyleBox	grabber
StyleBox	grabber_highlight
StyleBox	grabber_pressed
StyleBox	scroll
StyleBox	scroll_focus

Signals

scrolling() 🔗

Emitted when the scrollbar is being scrolled.

Property Descriptions

float **custom_step** = `-1.0`

- void **set_custom_step**(value: float)
- float **get_custom_step**()

Overrides the step used when clicking increment and decrement buttons or when using arrow keys when the **ScrollBar** is focused.

Theme Property Descriptions

Texture2D **decrement**

Icon used as a button to scroll the **ScrollBar** left/up. Supports custom step using the custom_step property.

Texture2D **decrement_highlight**

Displayed when the mouse cursor hovers over the decrement button.

Texture2D **decrement_pressed**

Displayed when the decrement button is being pressed.

Texture2D **increment**

Icon used as a button to scroll the **ScrollBar** right/down. Supports custom step using the custom_step property.

Texture2D **increment_highlight**

Displayed when the mouse cursor hovers over the increment button.

Texture2D **increment_pressed**

Displayed when the increment button is being pressed.

StyleBox **grabber**

Used as texture for the grabber, the draggable element representing current scroll.

StyleBox **grabber_highlight**

Used when the mouse hovers over the grabber.

StyleBox **grabber_pressed**

Used when the grabber is being dragged.

StyleBox **scroll**

Used as background of this **ScrollBar**.

StyleBox **scroll_focus**

Used as background when the **ScrollBar** has the GUI focus.

ScrollContainer

Inherits: Container **<** Control **<** CanvasItem **<** Node **<** Object

Inherited By: EditorInspector

A container used to provide scrollbars to a child control when needed.

Description

A container used to provide a child control with scrollbars when needed. Scrollbars will automatically be drawn at the right (for vertical) or bottom (for horizontal) and will enable dragging to move the viewable Control (and its children) within the ScrollContainer. Scrollbars will also automatically resize the grabber based on the Control.custom_minimum_size of the Control relative to the ScrollContainer.

Tutorials

- Using Containers

Properties

bool	clip_contents	`true` (overrides Control)
bool	follow_focus	`false`

ScrollMode	horizontal_scroll_mode	`1`
int	scroll_deadzone	`0`
int	scroll_horizontal	`0`
float	scroll_horizontal_custom_step	`-1.0`
int	scroll_vertical	`0`
float	scroll_vertical_custom_step	`-1.0`
ScrollMode	vertical_scroll_mode	`1`

Methods

void	ensure_control_visible(control: Control)
HScrollBar	get_h_scroll_bar()
VScrollBar	get_v_scroll_bar()

Theme Properties

StyleBox panel

Signals

scroll_ended() 🔗

Emitted when scrolling stops when dragging the scrollable area *with a touch event*. This signal is *not* emitted when scrolling by dragging the scrollbar, scrolling with the mouse wheel or scrolling with keyboard/gamepad events.

Note: This signal is only emitted on Android or iOS, or on desktop/web platforms when ProjectSettings.input_devices/pointing/emulate_touch_from_mouse is enabled.

---

scroll_started() 🔗

Emitted when scrolling starts when dragging the scrollable area w*ith a touch event*. This signal is *not* emitted when scrolling by dragging the scrollbar, scrolling with the mouse wheel or scrolling with keyboard/gamepad events.

Note: This signal is only emitted on Android or iOS, or on desktop/web platforms when ProjectSettings.input_devices/pointing/emulate_touch_from_mouse is enabled.

Enumerations

enum **ScrollMode**: 🔗

ScrollMode **SCROLL_MODE_DISABLED** = `0`

Scrolling disabled, scrollbar will be invisible.

ScrollMode **SCROLL_MODE_AUTO** = `1`

Scrolling enabled, scrollbar will be visible only if necessary, i.e. container's content is bigger than the container.

ScrollMode **SCROLL_MODE_SHOW_ALWAYS** = `2`

Scrolling enabled, scrollbar will be always visible.

ScrollMode **SCROLL_MODE_SHOW_NEVER** = `3`

Scrolling enabled, scrollbar will be hidden.

Property Descriptions

bool **follow_focus** = `false` 🔗

- void **set_follow_focus**(value: bool)
- bool **is_following_focus**()

If `true`, the ScrollContainer will automatically scroll to focused children (including indirect children) to make sure they are fully visible.

ScrollMode **horizontal_scroll_mode** = `1` 🔗

- void **set_horizontal_scroll_mode**(value: ScrollMode)
- ScrollMode **get_horizontal_scroll_mode**()

Controls whether horizontal scrollbar can be used and when it should be visible. See ScrollMode for options.

---

int **scroll_deadzone** = `0` 🔗

- void **set_deadzone**(value: int)
- int **get_deadzone**()

Deadzone for touch scrolling. Lower deadzone makes the scrolling more sensitive.

---

int **scroll_horizontal** = `0` 🔗

- void **set_h_scroll**(value: int)
- int **get_h_scroll**()

The current horizontal scroll value.

Note: If you are setting this value in the Node._ready function or earlier, it needs to be wrapped with Object.set_deferred, since scroll bar's Range.max_value is not initialized yet.

```
func _ready():
    set_deferred("scroll_horizontal", 600)
```

---

float **scroll_horizontal_custom_step** = `-1.0` 🔗

- void **set_horizontal_custom_step**(value: float)
- float **get_horizontal_custom_step**()

Overrides the ScrollBar.custom_step used when clicking the internal scroll bar's horizontal increment and decrement buttons or when using arrow keys when the ScrollBar is focused.

---

int **scroll_vertical** = `0` 🔗

- void **set_v_scroll**(value: int)
- int **get_v_scroll**()

The current vertical scroll value.

Note: Setting it early needs to be deferred, just like in scroll_horizontal.

```
func _ready():
    set_deferred("scroll_vertical", 600)
```

---

float **scroll_vertical_custom_step** = `-1.0` 🔗

- void **set_vertical_custom_step**(value: float)
- float **get_vertical_custom_step**()

Overrides the ScrollBar.custom_step used when clicking the internal scroll bar's vertical increment and decrement buttons or when using arrow keys when the ScrollBar is focused.

ScrollMode **vertical_scroll_mode** = `1` 🔗

- void **set_vertical_scroll_mode**(value: ScrollMode)
- ScrollMode **get_vertical_scroll_mode**()

Controls whether vertical scrollbar can be used and when it should be visible. See ScrollMode for options.

Method Descriptions

void **ensure_control_visible**(control: Control) 🔗

Ensures the given `control` is visible (must be a direct or indirect child of the ScrollContainer). Used by follow_focus.

Note: This will not work on a node that was just added during the same frame. If you want to scroll to a newly added child, you must wait until the next frame using SceneTree.process_frame:

```
add_child(child_node)
await get_tree().process_frame
ensure_control_visible(child_node)
```

HScrollBar **get_h_scroll_bar**() 🔗

Returns the horizontal scrollbar HScrollBar of this **ScrollContainer**.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to disable or hide a scrollbar, you can use horizontal_scroll_mode.

VScrollBar **get_v_scroll_bar**() 🔗

Returns the vertical scrollbar VScrollBar of this **ScrollContainer**.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to disable or hide a scrollbar, you can use vertical_scroll_mode.

Theme Property Descriptions

StyleBox **panel** 🔗

The background StyleBox of the **ScrollContainer**.

Separator

Inherits: Control **<** CanvasItem **<** Node **<** Object

Inherited By: HSeparator, VSeparator

Abstract base class for separators.

Description

Abstract base class for separators, used for separating other controls. **Separator**s are purely visual and normally drawn as a StyleBoxLine.

Theme Properties

int	separation	0
StyleBox	separator	

Theme Property Descriptions

int **separation** = 0 🔗

The size of the area covered by the separator. Effectively works like a minimum width/height.

StyleBox **separator** 🔗

The style for the separator line. Works best with StyleBoxLine (remember to enable StyleBoxLine.vertical for VSeparator).

ShaderGlobalsOverride

Inherits: Node **<** Object

A node used to override global shader parameters' values in a scene.

Description

Similar to how a WorldEnvironment node can be used to override the environment while a specific scene is loaded, **ShaderGlobalsOverride** can be used to override global shader parameters temporarily. Once the node is removed, the project-wide values for the global shader parameters are restored. See the RenderingServer `global_shader_parameter_*` methods for more information.

Note: Only one **ShaderGlobalsOverride** can be used per scene. If there is more than one **ShaderGlobalsOverride** node in the scene tree, only the first node (in tree order) will be taken into account.

Note: All **ShaderGlobalsOverride** nodes are made part of a `"shader_overrides_group"` group when they are added to the scene tree. The currently active **ShaderGlobalsOverride** node also has a `"shader_overrides_group_active"` group added to it. You can use this to check which **ShaderGlobalsOverride** node is currently active.

Tutorials

- Shading language

ShapeCast2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

A 2D shape that sweeps a region of space to detect CollisionObject2Ds.

Description

Shape casting allows to detect collision objects by sweeping its shape along the cast direction determined by target_position. This is similar to RayCast2D, but it allows for sweeping a region of space, rather than just a straight line. **ShapeCast2D** can detect multiple collision objects. It is useful for things like wide laser beams or snapping a simple shape to a floor.

Immediate collision overlaps can be done with the target_position set to `Vector2(0, 0)` and by calling force_shapecast_update within the same physics frame. This helps to overcome some limitations of Area2D when used as an instantaneous detection area, as collision information isn't immediately available to it.

Note: Shape casting is more computationally expensive than ray casting.

Properties

bool	collide_with_areas	`false`
bool	collide_with_bodies	`true`

int	collision_mask	`1`
Array	collision_result	`[]`
bool	enabled	`true`
bool	exclude_parent	`true`
float	margin	`0.0`
int	max_results	`32`
Shape2D	shape	
Vector2	target_position	`Vector2(0, 50)`

Methods

void	add_exception(node: CollisionObject2D)
void	add_exception_rid(rid: RID)
void	clear_exceptions()

void	force_shapecast_update()
float	get_closest_collision_safe_fraction() const
float	get_closest_collision_unsafe_fraction() const
Object	get_collider(index: int) const
RID	get_collider_rid(index: int) const
int	get_collider_shape(index: int) const
int	get_collision_count() const
bool	get_collision_mask_value(layer_number: int) const
Vector2	get_collision_normal(index: int) const
Vector2	get_collision_point(index: int) const
bool	is_colliding() const
void	remove_exception(node: CollisionObject2D)
void	remove_exception_rid(rid: RID)

void	set_collision_mask_value(layer_number: int, value: bool)

Property Descriptions

bool **collide_with_areas** = `false`

- void **set_collide_with_areas**(value: bool)
- bool **is_collide_with_areas_enabled**()

If `true`, collisions with Area2Ds will be reported.

bool **collide_with_bodies** = `true`

- void **set_collide_with_bodies**(value: bool)
- bool **is_collide_with_bodies_enabled**()

If `true`, collisions with PhysicsBody2Ds will be reported.

int **collision_mask** = `1`

- void **set_collision_mask**(value: int)
- int **get_collision_mask**()

The shape's collision mask. Only objects in at least one collision layer enabled in the mask will be detected.

Array **collision_result** = `[]`

Returns the complete collision information from the collision sweep. The data returned is the same as in the

PhysicsDirectSpaceState2D.get_rest_info method.

bool **enabled** = `true` 🔗

- void **set_enabled**(value: bool)
- bool **is_enabled**()

If `true`, collisions will be reported.

bool **exclude_parent** = `true` 🔗

- void **set_exclude_parent_body**(value: bool)
- bool **get_exclude_parent_body**()

If `true`, the parent node will be excluded from collision detection.

float **margin** = `0.0` 🔗

- void **set_margin**(value: float)
- float **get_margin**()

The collision margin for the shape. A larger margin helps detecting collisions more consistently, at the cost of precision.

int **max_results** = `32` 🔗

- void **set_max_results**(value: int)
- int **get_max_results**()

The number of intersections can be limited with this parameter, to reduce the processing time.

Shape2D **shape** 🔗

- void **set_shape**(value: Shape2D)
- Shape2D **get_shape**()

The Shape2D-derived shape to be used for collision queries.

Vector2 **target_position** = `Vector2(0, 50)` 🔗

- void **set_target_position**(value: Vector2)
- Vector2 **get_target_position**()

The shape's destination point, relative to this node's `position`.

Method Descriptions

void **add_exception**(node: CollisionObject2D) 🔗

Adds a collision exception so the shape does not report collisions with the specified CollisionObject2D node.

---

void **add_exception_rid**(rid: RID) 🔗

Adds a collision exception so the shape does not report collisions with the specified RID.

---

void **clear_exceptions**() 🔗

Removes all collision exceptions for this shape.

---

void **force_shapecast_update**() 🔗

Updates the collision information for the shape immediately, without waiting for the next `_physics_process` call. Use this method, for example, when the shape or its parent has changed state.

Note: `enabled == true` is not required for this to work.

---

float **get_closest_collision_safe_fraction**() const

The fraction from the **ShapeCast2D**'s origin to its target_position (between 0 and 1) of how far the shape can move without triggering a collision.

---

float **get_closest_collision_unsafe_fraction**() const

The fraction from the **ShapeCast2D**'s origin to its target_position (between 0 and 1) of how far the shape must move to trigger a collision.

In ideal conditions this would be the same as get_closest_collision_safe_fraction, however shape casting is calculated in discrete steps, so the precise point of collision can occur between two calculated positions.

Object **get_collider**(index: int) const

Returns the collided Object of one of the multiple collisions at `index`, or `null` if no object is intersecting the shape (i.e. is_colliding returns `false`).

RID **get_collider_rid**(index: int) const

Returns the RID of the collided object of one of the multiple collisions at `index`.

---

int **get_collider_shape**(index: int) const

Returns the shape ID of the colliding shape of one of the multiple collisions at `index`, or `0` if no object is intersecting the shape (i.e. is_colliding returns `false`).

---

int **get_collision_count**() const

The number of collisions detected at the point of impact. Use this to iterate over multiple collisions as provided by get_collider, get_collider_shape, get_collision_point, and get_collision_normal methods.

---

bool **get_collision_mask_value**(layer_number: int) const

Returns whether or not the specified layer of the collision_mask is enabled, given a `layer_number` between 1 and 32.

Vector2 **get_collision_normal**(index: int) const

Returns the normal of one of the multiple collisions at `index` of the intersecting object.

Vector2 **get_collision_point**(index: int) const

Returns the collision point of one of the multiple collisions at `index` where the shape intersects the colliding object.

Note: this point is in the **global** coordinate system.

bool **is_colliding**() const

Returns whether any object is intersecting with the shape's vector (considering the vector length).

void **remove_exception**(node: CollisionObject2D)

Removes a collision exception so the shape does report collisions with the specified CollisionObject2D node.

void **remove_exception_rid**(rid: RID)

Removes a collision exception so the shape does report collisions with the specified RID.

void **set_collision_mask_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the collision_mask, given a `layer_number` between 1 and 32.

ShapeCast3D

Inherits: Node3D **<** Node **<** Object

A 3D shape that sweeps a region of space to detect CollisionObject3Ds.

Description

Shape casting allows to detect collision objects by sweeping its shape along the cast direction determined by target_position. This is similar to RayCast3D, but it allows for sweeping a region of space, rather than just a straight line. **ShapeCast3D** can detect multiple collision objects. It is useful for things like wide laser beams or snapping a simple shape to a floor.

Immediate collision overlaps can be done with the target_position set to `Vector3(0, 0, 0)` and by calling force_shapecast_update within the same physics frame. This helps to overcome some limitations of Area3D when used as an instantaneous detection area, as collision information isn't immediately available to it.

Note: Shape casting is more computationally expensive than ray casting.

Properties

bool	collide_with_areas	`false`
bool	collide_with_bodies	`true`

int	collision_mask	`1`
Array	collision_result	`[]`
Color	debug_shape_custom_color	`Color(0, 0, 0, 1)`
bool	enabled	`true`
bool	exclude_parent	`true`
float	margin	`0.0`
int	max_results	`32`
Shape3D	shape	
Vector3	target_position	`Vector3(0, -1, 0)`

Methods

void	add_exception(node: CollisionObject3D)
void	add_exception_rid(rid: RID)

void	clear_exceptions()
void	force_shapecast_update()
float	get_closest_collision_safe_fraction() const
float	get_closest_collision_unsafe_fraction() const
Object	get_collider(index: int) const
RID	get_collider_rid(index: int) const
int	get_collider_shape(index: int) const
int	get_collision_count() const
bool	get_collision_mask_value(layer_number: int) const
Vector3	get_collision_normal(index: int) const
Vector3	get_collision_point(index: int) const
bool	is_colliding() const
void	remove_exception(node: CollisionObject3D)

void	remove_exception_rid(rid: RID)
void	resource_changed(resource: Resource)
void	set_collision_mask_value(layer_number: int, value: bool)

Property Descriptions

bool **collide_with_areas** = `false` 🔗

- void **set_collide_with_areas**(value: bool)
- bool **is_collide_with_areas_enabled**()

If `true`, collisions with Area3Ds will be reported.

bool **collide_with_bodies** = `true` 🔗

- void **set_collide_with_bodies**(value: bool)
- bool **is_collide_with_bodies_enabled**()

If `true`, collisions with PhysicsBody3Ds will be reported.

int **collision_mask** = `1` 🔗

- void **set_collision_mask**(value: int)
- int **get_collision_mask**()

The shape's collision mask. Only objects in at least one collision layer enabled in the mask will be detected. See Collision layers and masks in the documentation for more information.

Array **collision_result** = `[]` 🔗

Returns the complete collision information from the collision sweep. The data returned is the same as in the PhysicsDirectSpaceState3D.get_rest_info method.

Color **debug_shape_custom_color** = `Color(0, 0, 0, 1)` 🔗

- void **set_debug_shape_custom_color**(value: Color)
- Color **get_debug_shape_custom_color**()

The custom color to use to draw the shape in the editor and at run-time if **Visible Collision Shapes** is enabled in the **Debug** menu. This color will be highlighted at run-time if the **ShapeCast3D** is colliding with something.

If set to `Color(0.0, 0.0, 0.0)` (by default), the color set in ProjectSettings.debug/shapes/collision/shape_color is used.

---

bool **enabled** = `true` 🔗

- void **set_enabled**(value: bool)
- bool **is_enabled**()

If `true`, collisions will be reported.

---

bool **exclude_parent** = `true` 🔗

- void **set_exclude_parent_body**(value: bool)
- bool **get_exclude_parent_body**()

If `true`, the parent node will be excluded from collision detection.

float **margin** = `0.0`

- void **set_margin**(value: float)
- float **get_margin**()

The collision margin for the shape. A larger margin helps detecting collisions more consistently, at the cost of precision.

int **max_results** = `32`

- void **set_max_results**(value: int)
- int **get_max_results**()

The number of intersections can be limited with this parameter, to reduce the processing time.

Shape3D **shape**

- void **set_shape**(value: Shape3D)
- Shape3D **get_shape**()

The Shape3D-derived shape to be used for collision queries.

Vector3 **target_position** = `Vector3(0, -1, 0)`

- void **set_target_position**(value: Vector3)
- Vector3 **get_target_position**()

The shape's destination point, relative to this node's `position`.

Method Descriptions

void **add_exception**(node: CollisionObject3D)

Adds a collision exception so the shape does not report collisions with the specified CollisionObject3D node.

---

void **add_exception_rid**(rid: RID)

Adds a collision exception so the shape does not report collisions with the specified RID.

---

void **clear_exceptions**()

Removes all collision exceptions for this **ShapeCast3D**.

---

void **force_shapecast_update**()

Updates the collision information for the shape immediately, without waiting for the next `_physics_process` call. Use this method, for example, when the shape or its parent has changed state.

Note: `enabled == true` is not required for this to work.

---

float **get_closest_collision_safe_fraction**() const

The fraction from the **ShapeCast3D**'s origin to its target_position (between 0 and 1) of how far the shape can move without triggering a collision.

---

float **get_closest_collision_unsafe_fraction**() const

The fraction from the **ShapeCast3D**'s origin to its target_position (between 0 and 1) of how far the shape must move to trigger a collision.

In ideal conditions this would be the same as get_closest_collision_safe_fraction, however shape casting is calculated in discrete steps, so the precise point of collision can occur between two calculated positions.

Object **get_collider**(index: int) const 🔗

Returns the collided Object of one of the multiple collisions at `index`, or `null` if no object is intersecting the shape (i.e. is_colliding returns `false`).

RID **get_collider_rid**(index: int) const 🔗

Returns the RID of the collided object of one of the multiple collisions at `index`.

---

int **get_collider_shape**(index: int) const 🔗

Returns the shape ID of the colliding shape of one of the multiple collisions at `index`, or `0` if no object is intersecting the shape (i.e. is_colliding returns `false`).

---

int **get_collision_count**() const 🔗

The number of collisions detected at the point of impact. Use this to iterate over multiple collisions as provided by get_collider, get_collider_shape, get_collision_point, and get_collision_normal methods.

---

bool **get_collision_mask_value**(layer_number: int) const

Returns whether or not the specified layer of the collision_mask is enabled, given a `layer_number` between 1 and 32.

Vector3 **get_collision_normal**(index: int) const

Returns the normal of one of the multiple collisions at `index` of the intersecting object.

Vector3 **get_collision_point**(index: int) const

Returns the collision point of one of the multiple collisions at `index` where the shape intersects the colliding object.

Note: this point is in the **global** coordinate system.

---

bool **is_colliding**() const

Returns whether any object is intersecting with the shape's vector (considering the vector length).

---

void **remove_exception**(node: CollisionObject3D)

Removes a collision exception so the shape does report collisions with the specified CollisionObject3D node.

---

void **remove_exception_rid**(rid: RID)

Removes a collision exception so the shape does report collisions with the specified RID.

---

void **resource_changed**(resource: Resource)

Deprecated: Use Resource.changed instead.

This method does nothing.

---

void **set_collision_mask_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the collision_mask, given a `layer_number` between 1 and 32.

Skeleton2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

The parent of a hierarchy of Bone2Ds, used to create a 2D skeletal animation.

Description

Skeleton2D parents a hierarchy of Bone2D nodes. It holds a reference to each Bone2D's rest pose and acts as a single point of access to its bones.

To set up different types of inverse kinematics for the given Skeleton2D, a SkeletonModificationStack2D should be created. The inverse kinematics be applied by increasing SkeletonModificationStack2D.modification_count and creating the desired number of modifications.

Tutorials

- 2D skeletons

Methods

void	execute_modifications(delta: float, execution_mode: int)
Bone2D	get_bone(idx: int)
int	get_bone_count() const

Transform2D	get_bone_local_pose_override(bone_idx: int)
SkeletonModificationStack2D	get_modification_stack() const
RID	get_skeleton() const
void	set_bone_local_pose_override(bone_idx: int, override_pose: Transform2D, strength: float, persistent: bool)
void	set_modification_stack(modification_stack: SkeletonModificationStack2D)

Signals

bone_setup_changed() 🔗

Emitted when the Bone2D setup attached to this skeletons changes. This is primarily used internally within the skeleton.

Method Descriptions

void **execute_modifications**(delta: float, execution_mode: int) 🔗

Executes all the modifications on the SkeletonModificationStack2D, if the Skeleton2D has one assigned.

Bone2D **get_bone**(idx: int)

Returns a Bone2D from the node hierarchy parented by Skeleton2D. The object to return is identified by the parameter `idx`. Bones are indexed by descending the node hierarchy from top to bottom, adding the children of each branch before moving to the next sibling.

---

int **get_bone_count**() const

Returns the number of Bone2D nodes in the node hierarchy parented by Skeleton2D.

Transform2D **get_bone_local_pose_override**(bone_idx: int)

Returns the local pose override transform for `bone_idx`.

SkeletonModificationStack2D **get_modification_stack**() const

Returns the SkeletonModificationStack2D attached to this skeleton, if one exists.

RID **get_skeleton**() const

Returns the RID of a Skeleton2D instance.

---

void **set_bone_local_pose_override**(bone_idx: int, override_pose: Transform2D, strength: float, persistent: bool)

Sets the local pose transform, `override_pose`, for the bone at `bone_idx`.

`strength` is the interpolation strength that will be used when applying the pose, and `persistent` determines if the applied pose will remain.

Note: The pose transform needs to be a local transform relative to the Bone2D node at `bone_idx`!

void **set_modification_stack**(modification_stack: SkeletonModificationStack2D) 🔗

Sets the SkeletonModificationStack2D attached to this skeleton.

Skeleton3D

Inherits: Node3D **<** Node **<** Object

A node containing a bone hierarchy, used to create a 3D skeletal animation.

Description

Skeleton3D provides an interface for managing a hierarchy of bones, including pose, rest and animation (see Animation). It can also use ragdoll physics.

The overall transform of a bone with respect to the skeleton is determined by bone pose. Bone rest defines the initial transform of the bone pose.

Note that "global pose" below refers to the overall transform of the bone with respect to skeleton, so it is not the actual global/world transform of the bone.

Tutorials

- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

bool	animate_physical_bones	`true`
ModifierCallbackModeProcess	modifier_callback_mode_process	`1`

float	motion_scale	`1.0`
bool	show_rest_only	`false`

Methods

int	add_bone(name: String)
void	clear_bones()
void	clear_bones_global_pose_override()
Skin	create_skin_from_rest_transforms()
int	find_bone(name: String) const
void	force_update_all_bone_transforms()
void	force_update_bone_child_transform(bone_idx: int)
PackedInt32Array	get_bone_children(bone_idx: int) const
int	get_bone_count() const
Transform3D	get_bone_global_pose(bone_idx: int) const

Transform3D	get_bone_global_pose_no_override(bone_idx: int) const
Transform3D	get_bone_global_pose_override(bone_idx: int) const
Transform3D	get_bone_global_rest(bone_idx: int) const
String	get_bone_name(bone_idx: int) const
int	get_bone_parent(bone_idx: int) const
Transform3D	get_bone_pose(bone_idx: int) const
Vector3	get_bone_pose_position(bone_idx: int) const
Quaternion	get_bone_pose_rotation(bone_idx: int) const
Vector3	get_bone_pose_scale(bone_idx: int) const
Transform3D	get_bone_rest(bone_idx: int) const
StringName	get_concatenated_bone_names() const
PackedInt32Array	get_parentless_bones() const
int	get_version() const

bool	is_bone_enabled(bone_idx: int) const
void	localize_rests()
void	physical_bones_add_collision_exception(exception: RID)
void	physical_bones_remove_collision_exception(exception: RID)
void	physical_bones_start_simulation(bones: Array[StringName] = [])
void	physical_bones_stop_simulation()
SkinReference	register_skin(skin: Skin)
void	reset_bone_pose(bone_idx: int)
void	reset_bone_poses()
void	set_bone_enabled(bone_idx: int, enabled: bool = true)
void	set_bone_global_pose(bone_idx: int, pose: Transform3D)
void	set_bone_global_pose_override(bone_idx: int, pose: Transform3D, amount: float, persistent: bool = false)

void	set_bone_name(bone_idx: int, name: String)
void	set_bone_parent(bone_idx: int, parent_idx: int)
void	set_bone_pose(bone_idx: int, pose: Transform3D)
void	set_bone_pose_position(bone_idx: int, position: Vector3)
void	set_bone_pose_rotation(bone_idx: int, rotation: Quaternion)
void	set_bone_pose_scale(bone_idx: int, scale: Vector3)
void	set_bone_rest(bone_idx: int, rest: Transform3D)
void	unparent_bone_and_rest(bone_idx: int)

Signals

bone_enabled_changed(bone_idx: int) 🔗

Emitted when the bone at `bone_idx` is toggled with set_bone_enabled. Use is_bone_enabled to check the new value.

bone_list_changed() 🔗

There is currently no description for this signal. Please help us by contributing one!

---

pose_updated()

Emitted when the pose is updated.

Note: During the update process, this signal is not fired, so modification by SkeletonModifier3D is not detected.

---

show_rest_only_changed()

Emitted when the value of show_rest_only changes.

---

skeleton_updated()

Emitted when the final pose has been calculated will be applied to the skin in the update process.

This means that all SkeletonModifier3D processing is complete. In order to detect the completion of the processing of each SkeletonModifier3D, use SkeletonModifier3D.modification_processed.

Enumerations

enum **ModifierCallbackModeProcess**:

ModifierCallbackModeProcess **MODIFIER_CALLBACK_MODE_PROCESS_PHYSICS** = `0`

Set a flag to process modification during physics frames (see Node.NOTIFICATION_INTERNAL_PHYSICS_PROCESS).

ModifierCallbackModeProcess **MODIFIER_CALLBACK_MODE_PROCESS_IDLE** = `1`

Set a flag to process modification during process frames (see Node.NOTIFICATION_INTERNAL_PROCESS).

Constants

NOTIFICATION_UPDATE_SKELETON = `50` 🔗

Notification received when this skeleton's pose needs to be updated. In that case, this is called only once per frame in a deferred process.

Property Descriptions

bool **animate_physical_bones** = `true` 🔗

- void **set_animate_physical_bones**(value: bool)
- bool **get_animate_physical_bones**()

Deprecated: This property may be changed or removed in future versions.

If you follow the recommended workflow and explicitly have PhysicalBoneSimulator3D as a child of **Skeleton3D**, you can control whether it is affected by raycasting without running physical_bones_start_simulation, by its SkeletonModifier3D.active.

However, for old (deprecated) configurations, **Skeleton3D** has an internal virtual PhysicalBoneSimulator3D for compatibility. This property controls the internal virtual PhysicalBoneSimulator3D's SkeletonModifier3D.active.

ModifierCallbackModeProcess **modifier_callback_mode_process** = `1` 🔗

- void **set_modifier_callback_mode_process**(value: ModifierCallbackModeProcess)

- ModifierCallbackModeProcess **get_modifier_callback_mode_process**()

Sets the processing timing for the Modifier.

---

float **motion_scale** = `1.0` 🔗

- void **set_motion_scale**(value: float)
- float **get_motion_scale**()

Multiplies the 3D position track animation.

Note: Unless this value is `1.0`, the key value in animation will not match the actual position value.

---

bool **show_rest_only** = `false` 🔗

- void **set_show_rest_only**(value: bool)
- bool **is_show_rest_only**()

If `true`, forces the bones in their default rest pose, regardless of their values. In the editor, this also prevents the bones from being edited.

Method Descriptions

int **add_bone**(name: String) 🔗

Adds a new bone with the given name. Returns the new bone's index, or `-1` if this method fails.

Note: Bone names should be unique, non empty, and cannot include the `:` and `/` characters.

---

void **clear_bones**() 🔗

Clear all the bones in this skeleton.

---

void **clear_bones_global_pose_override**() 🔗

Deprecated: This method may be changed or removed in future versions.

Removes the global pose override on all bones in the skeleton.

Skin **create_skin_from_rest_transforms**() 🔗

There is currently no description for this method. Please help us by contributing one!

---

int **find_bone**(name: String) const 🔗

Returns the bone index that matches `name` as its name. Returns `-1` if no bone with this name exists.

---

void **force_update_all_bone_transforms**() 🔗

Deprecated: This method should only be called internally.

Force updates the bone transforms/poses for all bones in the skeleton.

---

void **force_update_bone_child_transform**(bone_idx: int) 🔗

Force updates the bone transform for the bone at `bone_idx` and all of its children.

PackedInt32Array **get_bone_children**(bone_idx: int) const 🔗

Returns an array containing the bone indexes of all the child node of the passed in bone, `bone_idx`.

int **get_bone_count**() const

Returns the number of bones in the skeleton.

Transform3D **get_bone_global_pose**(bone_idx: int) const

Returns the overall transform of the specified bone, with respect to the skeleton. Being relative to the skeleton frame, this is not the actual "global" transform of the bone.

Note: This is the global pose you set to the skeleton in the process, the final global pose can get overridden by modifiers in the deferred process, if you want to access the final global pose, use SkeletonModifier3D.modification_processed.

Transform3D **get_bone_global_pose_no_override**(bone_idx: int) const

Deprecated: This method may be changed or removed in future versions.

Returns the overall transform of the specified bone, with respect to the skeleton, but without any global pose overrides. Being relative to the skeleton frame, this is not the actual "global" transform of the bone.

Transform3D **get_bone_global_pose_override**(bone_idx: int) const

Deprecated: This method may be changed or removed in future versions.

Returns the global pose override transform for `bone_idx`.

Transform3D **get_bone_global_rest**(bone_idx: int) const

Returns the global rest transform for `bone_idx`.

String **get_bone_name**(bone_idx: int) const

Returns the name of the bone at index `bone_idx`.

---

int **get_bone_parent**(bone_idx: int) const

Returns the bone index which is the parent of the bone at `bone_idx`. If -1, then bone has no parent.

Note: The parent bone returned will always be less than `bone_idx`.

Transform3D **get_bone_pose**(bone_idx: int) const

Returns the pose transform of the specified bone.

Note: This is the pose you set to the skeleton in the process, the final pose can get overridden by modifiers in the deferred process, if you want to access the final pose, use SkeletonModifier3D.modification_processed.

Vector3 **get_bone_pose_position**(bone_idx: int) const

Returns the pose position of the bone at `bone_idx`. The returned Vector3 is in the local coordinate space of the **Skeleton3D** node.

Quaternion **get_bone_pose_rotation**(bone_idx: int) const

Returns the pose rotation of the bone at `bone_idx`. The returned Quaternion is local to the bone with respect to the rotation of any parent bones.

Vector3 **get_bone_pose_scale**(bone_idx: int) const

Returns the pose scale of the bone at `bone_idx`.

Transform3D **get_bone_rest**(bone_idx: int) const 🔗

Returns the rest transform for a bone `bone_idx`.

StringName **get_concatenated_bone_names**() const 🔗

Returns all bone names concatenated with commas (`,`) as a single StringName.

It is useful to set it as a hint for the enum property.

PackedInt32Array **get_parentless_bones**() const 🔗

Returns an array with all of the bones that are parentless. Another way to look at this is that it returns the indexes of all the bones that are not dependent or modified by other bones in the Skeleton.

---

int **get_version**() const 🔗

Returns the number of times the bone hierarchy has changed within this skeleton, including renames.

The Skeleton version is not serialized: only use within a single instance of Skeleton3D.

Use for invalidating caches in IK solvers and other nodes which process bones.

---

bool **is_bone_enabled**(bone_idx: int) const 🔗

Returns whether the bone pose for the bone at `bone_idx` is enabled.

---

void **localize_rests**()

Returns all bones in the skeleton to their rest poses.

---

void **physical_bones_add_collision_exception**(exception: RID)

Deprecated: This method may be changed or removed in future versions.

Adds a collision exception to the physical bone.

Works just like the RigidBody3D node.

---

void **physical_bones_remove_collision_exception**(exception: RID)

Deprecated: This method may be changed or removed in future versions.

Removes a collision exception to the physical bone.

Works just like the RigidBody3D node.

---

void **physical_bones_start_simulation**(bones: Array[StringName] = [])

Deprecated: This method may be changed or removed in future versions.

Tells the PhysicalBone3D nodes in the Skeleton to start simulating and reacting to the physics world.

Optionally, a list of bone names can be passed-in, allowing only the passed-in bones to be simulated.

---

void **physical_bones_stop_simulation**()

Deprecated: This method may be changed or removed in future versions.

Tells the PhysicalBone3D nodes in the Skeleton to stop simulating.

SkinReference **register_skin**(skin: Skin) 🔗

Binds the given Skin to the Skeleton.

---

void **reset_bone_pose**(bone_idx: int) 🔗

Sets the bone pose to rest for `bone_idx`.

---

void **reset_bone_poses**() 🔗

Sets all bone poses to rests.

---

void **set_bone_enabled**(bone_idx: int, enabled: bool = true) 🔗

Disables the pose for the bone at `bone_idx` if `false`, enables the bone pose if `true`.

---

void **set_bone_global_pose**(bone_idx: int, pose: Transform3D) 🔗

Sets the global pose transform, `pose`, for the bone at `bone_idx`.

Note: If other bone poses have been changed, this method executes a dirty poses recalculation and will cause performance to deteriorate. If you know that multiple global poses will be applied, consider using set_bone_pose with precalculation.

---

void **set_bone_global_pose_override**(bone_idx: int, pose: Transform3D, amount: float, persistent: bool = false) 🔗

Deprecated: This method may be changed or removed in future versions.

Sets the global pose transform, `pose`, for the bone at `bone_idx`.

`amount` is the interpolation strength that will be used when applying the pose, and `persistent` determines if the applied pose will remain.

Note: The pose transform needs to be a global pose! To convert a world transform from a Node3D to a global bone pose, multiply the Transform3D.affine_inverse of the node's Node3D.global_transform by the desired world transform.

void **set_bone_name**(bone_idx: int, name: String) 🔗

Sets the bone name, `name`, for the bone at `bone_idx`.

void **set_bone_parent**(bone_idx: int, parent_idx: int) 🔗

Sets the bone index `parent_idx` as the parent of the bone at `bone_idx`. If -1, then bone has no parent.

Note: `parent_idx` must be less than `bone_idx`.

void **set_bone_pose**(bone_idx: int, pose: Transform3D) 🔗

Sets the pose transform, `pose`, for the bone at `bone_idx`.

void **set_bone_pose_position**(bone_idx: int, position: Vector3) 🔗

Sets the pose position of the bone at `bone_idx` to `position`. `position` is a Vector3 describing a position local to the **Skeleton3D** node.

void **set_bone_pose_rotation**(bone_idx: int, rotation: Quaternion) 🔗

Sets the pose rotation of the bone at `bone_idx` to `rotation`. `rotation` is a Quaternion describing a rotation in the bone's local coordinate space with respect to the rotation of any parent bones.

void **set_bone_pose_scale**(bone_idx: int, scale: Vector3) 🔗

Sets the pose scale of the bone at `bone_idx` to `scale`.

void **set_bone_rest**(bone_idx: int, rest: Transform3D) 🔗

Sets the rest transform for bone `bone_idx`.

void **unparent_bone_and_rest**(bone_idx: int) 🔗

Unparents the bone at `bone_idx` and sets its rest position to that of its parent prior to being reset.

SkeletonIK3D

Deprecated: This class may be changed or removed in future versions.

Inherits: SkeletonModifier3D **<** Node3D **<** Node **<** Object

A node used to rotate all bones of a Skeleton3D bone chain a way that places the end bone at a desired 3D position.

Description

SkeletonIK3D is used to rotate all bones of a Skeleton3D bone chain a way that places the end bone at a desired 3D position. A typical scenario for IK in games is to place a character's feet on the ground or a character's hands on a currently held object. SkeletonIK uses FabrikInverseKinematic internally to solve the bone chain and applies the results to the Skeleton3D `bones_global_pose_override` property for all affected bones in the chain. If fully applied, this overwrites any bone transform from Animations or bone custom poses set by users. The applied amount can be controlled with the SkeletonModifier3D.influence property.

```
# Apply IK effect automatically on every new frame (not the
current)
skeleton_ik_node.start()

# Apply IK effect only on the current frame
skeleton_ik_node.start(true)

# Stop IK effect and reset bones_global_pose_override on
Skeleton
skeleton_ik_node.stop()

# Apply full IK effect
```

```
skeleton_ik_node.set_influence(1.0)

# Apply half IK effect
skeleton_ik_node.set_influence(0.5)

# Apply zero IK effect (a value at or below 0.01 also removes
bones_global_pose_override on Skeleton)
skeleton_ik_node.set_influence(0.0)
```

Properties

float	interpolation	
Vector3	magnet	`Vector3(0, 0, 0)`
int	max_iterations	`10`
float	min_distance	`0.01`
bool	override_tip_basis	`true`
StringName	root_bone	`&""`
Transform3D	target	`Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)`
NodePath	target_node	`NodePath("")`

StringName	tip_bone	`&""`
bool	use_magnet	`false`

Methods

Skeleton3D	get_parent_skeleton() const
bool	is_running()
void	start(one_time: bool = false)
void	stop()

Property Descriptions

float interpolation 🔗

- void **set_interpolation**(value: float)
- float **get_interpolation**()

Deprecated: Use SkeletonModifier3D.influence instead.

Interpolation value for how much the IK results are applied to the current skeleton bone chain. A value of `1.0` will overwrite all skeleton bone transforms completely while a value of `0.0` will visually disable the SkeletonIK.

Vector3 **magnet** = `Vector3(0, 0, 0)` 🔗

- void **set_magnet_position**(value: Vector3)
- Vector3 **get_magnet_position**()

Secondary target position (first is target property or target_node) for the IK chain. Use magnet position (pole target) to control the bending of the IK chain. Only works if the bone chain has more than 2 bones. The middle chain bone position will be linearly interpolated with the magnet position.

---

int **max_iterations** = `10` 🔗

- void **set_max_iterations**(value: int)
- int **get_max_iterations**()

Number of iteration loops used by the IK solver to produce more accurate (and elegant) bone chain results.

---

float **min_distance** = `0.01` 🔗

- void **set_min_distance**(value: float)
- float **get_min_distance**()

The minimum distance between bone and goal target. If the distance is below this value, the IK solver stops further iterations.

---

bool **override_tip_basis** = `true` 🔗

- void **set_override_tip_basis**(value: bool)

- bool **is_override_tip_basis**()

If `true` overwrites the rotation of the tip bone with the rotation of the target (or target_node if defined).

StringName **root_bone** = `&""`

- void **set_root_bone**(value: StringName)
- StringName **get_root_bone**()

The name of the current root bone, the first bone in the IK chain.

Transform3D **target** = `Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)`

- void **set_target_transform**(value: Transform3D)
- Transform3D **get_target_transform**()

First target of the IK chain where the tip bone is placed and, if override_tip_basis is `true`, how the tip bone is rotated. If a target_node path is available the nodes transform is used instead and this property is ignored.

NodePath **target_node** = `NodePath("")`

- void **set_target_node**(value: NodePath)
- NodePath **get_target_node**()

Target node NodePath for the IK chain. If available, the node's current Transform3D is used instead of the target property.

StringName **tip_bone** = `&""`

- void **set_tip_bone**(value: StringName)
- StringName **get_tip_bone**()

The name of the current tip bone, the last bone in the IK chain placed at the target transform (or target_node if defined).

---

bool **use_magnet** = `false` 🔗

- void **set_use_magnet**(value: bool)
- bool **is_using_magnet**()

If `true`, instructs the IK solver to consider the secondary magnet target (pole target) when calculating the bone chain. Use the magnet position (pole target) to control the bending of the IK chain.

Method Descriptions

Skeleton3D **get_parent_skeleton**() const 🔗

Returns the parent Skeleton3D Node that was present when SkeletonIK entered the SceneTree. Returns null if the parent node was not a Skeleton3D Node when SkeletonIK3D entered the SceneTree.

---

bool **is_running**() 🔗

Returns `true` if SkeletonIK is applying IK effects on continues frames to the Skeleton3D bones. Returns `false` if SkeletonIK is stopped or start was used with the `one_time` parameter set to `true`.

---

void **start**(one_time: bool = false) 🔗

Starts applying IK effects on each frame to the Skeleton3D bones but will only take effect starting on the next frame. If `one_time` is `true`, this will take effect immediately but also reset on the next frame.

void **stop**() 🔗

Stops applying IK effects on each frame to the Skeleton3D bones and also calls Skeleton3D.clear_bones_global_pose_override to remove existing overrides on all bones.

SkeletonModifier3D

Inherits: Node3D **<** Node **<** Object

Inherited By: PhysicalBoneSimulator3D, SkeletonIK3D, XRBodyModifier3D, XRHandModifier3D

A Node that may modify Skeleton3D's bone.

Description

SkeletonModifier3D retrieves a target Skeleton3D by having a Skeleton3D parent.

If there is AnimationMixer, modification always performs after playback process of the AnimationMixer.

This node should be used to implement custom IK solvers, constraints, or skeleton physics.

Tutorials

- Design of the Skeleton Modifier 3D [https://godotengine.org/article/design-of-the-skeleton-modifier-3d/]

Properties

bool	active	`true`

float	influence	`1.0`

Methods

void	_process_modification() virtual
Skeleton3D	get_skeleton() const

Signals

modification_processed() 🔗

Notifies when the modification have been finished.

Note: If you want to get the modified bone pose by the modifier, you must use Skeleton3D.get_bone_pose or Skeleton3D.get_bone_global_pose at the moment this signal is fired.

Property Descriptions

bool **active** = `true` 🔗

- void **set_active**(value: bool)
- bool **is_active**()

If `true`, the **SkeletonModifier3D** will be processing.

float **influence** = `1.0` 🔗

- void **set_influence**(value: float)
- float **get_influence**()

Sets the influence of the modification.

Note: This value is used by Skeleton3D to blend, so the **SkeletonModifier3D** should always apply only 100% of the result without interpolation.

Method Descriptions

void **_process_modification**() virtual 🔗

Override this virtual method to implement a custom skeleton modifier. You should do things like get the Skeleton3D's current pose and apply the pose here.

_process_modification must not apply influence to bone poses because the Skeleton3D automatically applies influence to all bone poses set by the modifier.

Skeleton3D **get_skeleton**() const 🔗

Get parent Skeleton3D node if found.

Slider

Inherits: Range **<** Control **<** CanvasItem **<** Node **<** Object

Inherited By: HSlider, VSlider

Abstract base class for sliders.

Description

Abstract base class for sliders, used to adjust a value by moving a grabber along a horizontal or vertical axis. Sliders are Range-based controls.

Properties

bool	editable	`true`
FocusMode	focus_mode	`2` (overrides Control)
bool	scrollable	`true`
float	step	`1.0` (overrides Range)
int	tick_count	`0`
bool	ticks_on_borders	`false`

Theme Properties

int	center_grabber	0
int	grabber_offset	0
Texture2D	grabber	
Texture2D	grabber_disabled	
Texture2D	grabber_highlight	
Texture2D	tick	
StyleBox	grabber_area	
StyleBox	grabber_area_highlight	
StyleBox	slider	

Signals

drag_ended(value_changed: bool) 🔗

Emitted when dragging stops. If `value_changed` is true, Range.value is different from the value when you started the dragging.

---

drag_started()

Emitted when dragging is started. This is emitted before the corresponding Range.value_changed signal.

Property Descriptions

bool **editable** = `true`

- void **set_editable**(value: bool)
- bool **is_editable**()

If `true`, the slider can be interacted with. If `false`, the value can be changed only by code.

---

bool **scrollable** = `true`

- void **set_scrollable**(value: bool)
- bool **is_scrollable**()

If `true`, the value can be changed using the mouse wheel.

---

int **tick_count** = `0`

- void **set_ticks**(value: int)
- int **get_ticks**()

Number of ticks displayed on the slider, including border ticks. Ticks are uniformly-distributed value markers.

---

bool **ticks_on_borders** = `false` 🔗

- void **set_ticks_on_borders**(value: bool)
- bool **get_ticks_on_borders**()

If `true`, the slider will display ticks for minimum and maximum values.

Theme Property Descriptions

int **center_grabber** = `0` 🔗

Boolean constant. If `1`, the grabber texture size will be ignored and it will fit within slider's bounds based only on its center position.

---

int **grabber_offset** = `0` 🔗

Vertical or horizontal offset of the grabber.

Texture2D **grabber** 🔗

The texture for the grabber (the draggable element).

Texture2D **grabber_disabled** 🔗

The texture for the grabber when it's disabled.

Texture2D **grabber_highlight** 🔗

The texture for the grabber when it's focused.

Texture2D **tick**

The texture for the ticks, visible when tick_count is greater than 0.

StyleBox **grabber_area**

The background of the area to the left or bottom of the grabber.

StyleBox **grabber_area_highlight**

The background of the area to the left or bottom of the grabber that displays when it's being hovered or focused.

StyleBox **slider**

The background for the whole slider. Affects the height or width of the grabber_area.

SliderJoint3D

Inherits: Joint3D **<** Node3D **<** Node **<** Object

A physics joint that restricts the movement of a 3D physics body along an axis relative to another physics body.

Description

A physics joint that restricts the movement of a 3D physics body along an axis relative to another physics body. For example, Body A could be a StaticBody3D representing a piston base, while Body B could be a RigidBody3D representing the piston head, moving up and down.

Properties

float	angular_limit/damping	`0.0`
float	angular_limit/lower_angle	`0.0`
float	angular_limit/restitution	`0.7`
float	angular_limit/softness	`1.0`
float	angular_limit/upper_angle	`0.0`

float	angular_motion/damping	`1.0`
float	angular_motion/restitution	`0.7`
float	angular_motion/softness	`1.0`
float	angular_ortho/damping	`1.0`
float	angular_ortho/restitution	`0.7`
float	angular_ortho/softness	`1.0`
float	linear_limit/damping	`1.0`
float	linear_limit/lower_distance	`-1.0`
float	linear_limit/restitution	`0.7`
float	linear_limit/softness	`1.0`
float	linear_limit/upper_distance	`1.0`
float	linear_motion/damping	`0.0`
float	linear_motion/restitution	`0.7`

float	linear_motion/softness	`1.0`
float	linear_ortho/damping	`1.0`
float	linear_ortho/restitution	`0.7`
float	linear_ortho/softness	`1.0`

Methods

float	get_param(param: Param) const
void	set_param(param: Param, value: float)

Enumerations

enum **Param**: 🔗

Param **PARAM_LINEAR_LIMIT_UPPER** = `0`

Constant for accessing linear_limit/upper_distance. The maximum difference between the pivot points on their X axis before damping happens.

Param **PARAM_LINEAR_LIMIT_LOWER** = `1`

Constant for accessing linear_limit/lower_distance. The minimum difference between the pivot points on their X axis before damping happens.

Param **PARAM_LINEAR_LIMIT_SOFTNESS** = `2`

Constant for accessing linear_limit/softness. A factor applied to the movement across the slider axis once the limits get surpassed. The lower, the slower the movement.

Param **PARAM_LINEAR_LIMIT_RESTITUTION** = `3`

Constant for accessing linear_limit/restitution. The amount of restitution once the limits are surpassed. The lower, the more velocity-energy gets lost.

Param **PARAM_LINEAR_LIMIT_DAMPING** = `4`

Constant for accessing linear_limit/damping. The amount of damping once the slider limits are surpassed.

Param **PARAM_LINEAR_MOTION_SOFTNESS** = `5`

Constant for accessing linear_motion/softness. A factor applied to the movement across the slider axis as long as the slider is in the limits. The lower, the slower the movement.

Param **PARAM_LINEAR_MOTION_RESTITUTION** = `6`

Constant for accessing linear_motion/restitution. The amount of restitution inside the slider limits.

Param **PARAM_LINEAR_MOTION_DAMPING** = `7`

Constant for accessing linear_motion/damping. The amount of damping inside the slider limits.

Param **PARAM_LINEAR_ORTHOGONAL_SOFTNESS** = `8`

Constant for accessing linear_ortho/softness. A factor applied to the movement across axes orthogonal to the slider.

Param **PARAM_LINEAR_ORTHOGONAL_RESTITUTION** = `9`

Constant for accessing linear_motion/restitution. The amount of restitution when movement is across axes orthogonal to the slider.

Param **PARAM_LINEAR_ORTHOGONAL_DAMPING** = `10`

Constant for accessing linear_motion/damping. The amount of damping when movement is across axes orthogonal to the slider.

Param **PARAM_ANGULAR_LIMIT_UPPER** = `11`

Constant for accessing angular_limit/upper_angle. The upper limit of rotation in the slider.

Param **PARAM_ANGULAR_LIMIT_LOWER** = `12`

Constant for accessing angular_limit/lower_angle. The lower limit of rotation in the slider.

Param **PARAM_ANGULAR_LIMIT_SOFTNESS** = `13`

Constant for accessing angular_limit/softness. A factor applied to the all rotation once the limit is surpassed.

Param **PARAM_ANGULAR_LIMIT_RESTITUTION** = `14`

Constant for accessing angular_limit/restitution. The amount of restitution of the rotation when the limit is surpassed.

Param **PARAM_ANGULAR_LIMIT_DAMPING** = `15`

Constant for accessing angular_limit/damping. The amount of damping of the rotation when the limit is surpassed.

Param **PARAM_ANGULAR_MOTION_SOFTNESS** = `16`

Constant for accessing angular_motion/softness. A factor applied to the all rotation in the limits.

Param **PARAM_ANGULAR_MOTION_RESTITUTION** = `17`

Constant for accessing angular_motion/restitution. The amount of restitution of the rotation in the limits.

Param **PARAM_ANGULAR_MOTION_DAMPING** = `18`

Constant for accessing angular_motion/damping. The amount of damping of the rotation in the limits.

Param **PARAM_ANGULAR_ORTHOGONAL_SOFTNESS** = `19`

Constant for accessing angular_ortho/softness. A factor applied to the all rotation across axes orthogonal to the slider.

Param **PARAM_ANGULAR_ORTHOGONAL_RESTITUTION** = `20`

Constant for accessing angular_ortho/restitution. The amount of restitution of the rotation across axes orthogonal to the slider.

Param **PARAM_ANGULAR_ORTHOGONAL_DAMPING** = `21`

Constant for accessing angular_ortho/damping. The amount of damping of the rotation across axes orthogonal to the slider.

Param **PARAM_MAX** = `22`

Represents the size of the Param enum.

Property Descriptions

float **angular_limit/damping** = `0.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The amount of damping of the rotation when the limit is surpassed.

A lower damping value allows a rotation initiated by body A to travel to body B slower.

---

float **angular_limit/lower_angle** = `0.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The lower limit of rotation in the slider.

---

float **angular_limit/restitution** = `0.7`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The amount of restitution of the rotation when the limit is surpassed.

Does not affect damping.

---

float **angular_limit/softness** = `1.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

A factor applied to the all rotation once the limit is surpassed.

Makes all rotation slower when between 0 and 1.

---

float **angular_limit/upper_angle** = `0.0` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The upper limit of rotation in the slider.

---

float **angular_motion/damping** = `1.0` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The amount of damping of the rotation in the limits.

---

float **angular_motion/restitution** = `0.7` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The amount of restitution of the rotation in the limits.

---

float **angular_motion/softness** = `1.0` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

A factor applied to the all rotation in the limits.

---

float **angular_ortho/damping** = `1.0` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The amount of damping of the rotation across axes orthogonal to the slider.

---

float **angular_ortho/restitution** = `0.7` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The amount of restitution of the rotation across axes orthogonal to the slider.

---

float **angular_ortho/softness** = `1.0` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

A factor applied to the all rotation across axes orthogonal to the slider.

---

float **linear_limit/damping** = `1.0` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The amount of damping that happens once the limit defined by linear_limit/lower_distance and linear_limit/upper_distance is surpassed.

---

float **linear_limit/lower_distance** = `-1.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The minimum difference between the pivot points on their X axis before damping happens.

---

float **linear_limit/restitution** = `0.7`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The amount of restitution once the limits are surpassed. The lower, the more velocity-energy gets lost.

---

float **linear_limit/softness** = `1.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

A factor applied to the movement across the slider axis once the limits get surpassed. The lower, the slower the movement.

---

float **linear_limit/upper_distance** = `1.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The maximum difference between the pivot points on their X axis before damping happens.

---

float **linear_motion/damping** = `0.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The amount of damping inside the slider limits.

---

float **linear_motion/restitution** = `0.7`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The amount of restitution inside the slider limits.

---

float **linear_motion/softness** = `1.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

A factor applied to the movement across the slider axis as long as the slider is in the limits. The lower, the slower the movement.

---

float **linear_ortho/damping** = `1.0`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The amount of damping when movement is across axes orthogonal to the slider.

---

float **linear_ortho/restitution** = `0.7`

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

The amount of restitution when movement is across axes orthogonal to the slider.

---

float **linear_ortho/softness** = `1.0` 🔗

- void **set_param**(param: Param, value: float)
- float **get_param**(param: Param) const

A factor applied to the movement across axes orthogonal to the slider.

Method Descriptions

float **get_param**(param: Param) const 🔗

Returns the value of the given parameter (see Param constants).

---

void **set_param**(param: Param, value: float) 🔗

Assigns `value` to the given parameter (see Param constants).

SoftBody3D

Inherits: MeshInstance3D **<** GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A deformable 3D physics mesh.

Description

A deformable 3D physics mesh. Used to create elastic or deformable objects such as cloth, rubber, or other flexible materials.

Additionally, **SoftBody3D** is subject to wind forces defined in Area3D (see Area3D.wind_source_path, Area3D.wind_force_magnitude, and Area3D.wind_attenuation_factor).

Note: There are many known bugs in **SoftBody3D**. Therefore, it's not recommended to use them for things that can affect gameplay (such as trampolines).

Tutorials

- SoftBody

Properties

int	collision_layer	`1`

int	collision_mask	`1`
float	damping_coefficient	`0.01`
DisableMode	disable_mode	`0`
float	drag_coefficient	`0.0`
float	linear_stiffness	`0.5`
NodePath	parent_collision_ignore	`NodePath("")`
float	pressure_coefficient	`0.0`
bool	ray_pickable	`true`
int	simulation_precision	`5`
float	total_mass	`1.0`

Methods

void	add_collision_exception_with(body: Node)

Array[PhysicsBody3D]	get_collision_exceptions()
bool	get_collision_layer_value(layer_number: int) const
bool	get_collision_mask_value(layer_number: int) const
RID	get_physics_rid() const
Vector3	get_point_transform(point_index: int)
bool	is_point_pinned(point_index: int) const
void	remove_collision_exception_with(body: Node)
void	set_collision_layer_value(layer_number: int, value: bool)
void	set_collision_mask_value(layer_number: int, value: bool)
void	set_point_pinned(point_index: int, pinned: bool, attachment_path: NodePath = NodePath(""))

Enumerations

enum **DisableMode**: 🔗

DisableMode **DISABLE_MODE_REMOVE** = `0`

When Node.process_mode is set to Node.PROCESS_MODE_DISABLED, remove from the physics simulation to stop all physics interactions with this **SoftBody3D**.

Automatically re-added to the physics simulation when the Node is processed again.

DisableMode **DISABLE_MODE_KEEP_ACTIVE** = `1`

When Node.process_mode is set to Node.PROCESS_MODE_DISABLED, do not affect the physics simulation.

Property Descriptions

int **collision_layer** = `1` 🔗

- void **set_collision_layer**(value: int)
- int **get_collision_layer**()

The physics layers this SoftBody3D **is in**. Collision objects can exist in one or more of 32 different layers. See also collision_mask.

Note: Object A can detect a contact with object B only if object B is in any of the layers that object A scans. See Collision layers and masks in the documentation for more information.

---

int **collision_mask** = `1`

- void **set_collision_mask**(value: int)
- int **get_collision_mask**()

The physics layers this SoftBody3D **scans**. Collision objects can scan one or more of 32 different layers. See also collision_layer.

Note: Object A can detect a contact with object B only if object B is in any of the layers that object A scans. See Collision layers and masks in the documentation for more information.

float **damping_coefficient** = `0.01`

- void **set_damping_coefficient**(value: float)
- float **get_damping_coefficient**()

The body's damping coefficient. Higher values will slow down the body more noticeably when forces are applied.

DisableMode **disable_mode** = `0`

- void **set_disable_mode**(value: DisableMode)
- DisableMode **get_disable_mode**()

Defines the behavior in physics when Node.process_mode is set to Node.PROCESS_MODE_DISABLED. See DisableMode for more details about the different modes.

float **drag_coefficient** = `0.0`

- void **set_drag_coefficient**(value: float)
- float **get_drag_coefficient**()

The body's drag coefficient. Higher values increase this body's air resistance.

Note: This value is currently unused by Godot's default physics implementation.

---

float **linear_stiffness** = `0.5`

- void **set_linear_stiffness**(value: float)
- float **get_linear_stiffness**()

Higher values will result in a stiffer body, while lower values will increase the body's ability to bend. The value can be between `0.0` and `1.0` (inclusive).

NodePath **parent_collision_ignore** = `NodePath("")`

- void **set_parent_collision_ignore**(value: NodePath)
- NodePath **get_parent_collision_ignore**()

NodePath to a CollisionObject3D this SoftBody3D should avoid clipping.

---

float **pressure_coefficient** = `0.0`

- void **set_pressure_coefficient**(value: float)
- float **get_pressure_coefficient**()

The pressure coefficient of this soft body. Simulate pressure build-up from inside this body. Higher values increase the strength of this effect.

---

bool **ray_pickable** = `true`

- void **set_ray_pickable**(value: bool)
- bool **is_ray_pickable**()

If `true`, the **SoftBody3D** will respond to RayCast3Ds.

int **simulation_precision** = `5`

- void **set_simulation_precision**(value: int)
- int **get_simulation_precision**()

Increasing this value will improve the resulting simulation, but can affect performance. Use with care.

float **total_mass** = `1.0`

- void **set_total_mass**(value: float)
- float **get_total_mass**()

The SoftBody3D's mass.

Method Descriptions

void **add_collision_exception_with**(body: Node)

Adds a body to the list of bodies that this body can't collide with.

Array[PhysicsBody3D] **get_collision_exceptions**()

Returns an array of nodes that were added as collision exceptions for this body.

bool **get_collision_layer_value**(layer_number: int) const

Returns whether or not the specified layer of the collision_layer is enabled, given a `layer_number` between 1 and 32.

---

bool **get_collision_mask_value**(layer_number: int) const

Returns whether or not the specified layer of the collision_mask is enabled, given a `layer_number` between 1 and 32.

RID **get_physics_rid**() const

Returns the internal RID used by the PhysicsServer3D for this body.

Vector3 **get_point_transform**(point_index: int)

Returns local translation of a vertex in the surface array.

---

bool **is_point_pinned**(point_index: int) const

Returns `true` if vertex is set to pinned.

---

void **remove_collision_exception_with**(body: Node)

Removes a body from the list of bodies that this body can't collide with.

---

void **set_collision_layer_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the collision_layer, given a `layer_number` between 1 and 32.

---

void **set_collision_mask_value**(layer_number: int, value: bool) 🔗

Based on `value`, enables or disables the specified layer in the collision_mask, given a `layer_number` between 1 and 32.

---

void **set_point_pinned**(point_index: int, pinned: bool, attachment_path: NodePath = NodePath("")) 🔗

Sets the pinned state of a surface vertex. When set to `true`, the optional `attachment_path` can define a Node3D the pinned vertex will be attached to.

SpinBox

Inherits: Range **<** Control **<** CanvasItem **<** Node **<** Object

An input field for numbers.

Description

SpinBox is a numerical input text field. It allows entering integers and floating-point numbers.

Example:

GDScript

```
var spin_box = SpinBox.new()
add_child(spin_box)
var line_edit = spin_box.get_line_edit()
line_edit.context_menu_enabled = false
spin_box.horizontal_alignment =
LineEdit.HORIZONTAL_ALIGNMENT_RIGHT
```

C#

```
var spinBox = new SpinBox();
AddChild(spinBox);
var lineEdit = spinBox.GetLineEdit();
lineEdit.ContextMenuEnabled = false;
spinBox.AlignHorizontal = LineEdit.HorizontalAlignEnum.Right;
```

The above code will create a **SpinBox**, disable context menu on it and set the text alignment to right.

See Range class for more options over the **SpinBox**.

Note: With the **SpinBox**'s context menu disabled, you can right-click the bottom half of the spinbox to set the value to its minimum, while right-clicking the top half sets the value to its maximum.

Note: SpinBox relies on an underlying LineEdit node. To theme a **SpinBox**'s background, add theme items for LineEdit and customize them.

Note: If you want to implement drag and drop for the underlying LineEdit, you can use Control.set_drag_forwarding on the node returned by get_line_edit.

Properties

HorizontalAlignment	alignment	`0`
float	custom_arrow_step	`0.0`
bool	editable	`true`
String	prefix	`""`
bool	select_all_on_focus	`false`
BitField[SizeFlags]	size_flags_vertical	`1` (overrides Control)
float	step	`1.0` (overrides Range)

String	suffix	`""`
bool	update_on_text_changed	`false`

Methods

void	apply()
LineEdit	get_line_edit()

Theme Properties

Texture2D	updown

Property Descriptions

HorizontalAlignment **alignment** = `0` 🔗

- void **set_horizontal_alignment**(value: HorizontalAlignment)
- HorizontalAlignment **get_horizontal_alignment**()

Changes the alignment of the underlying LineEdit.

---

float **custom_arrow_step** = `0.0` 🔗

- void **set_custom_arrow_step**(value: float)
- float **get_custom_arrow_step**()

If not `0`, Range.value will always be rounded to a multiple of custom_arrow_step when interacting with the arrow buttons of the **SpinBox**.

---

bool **editable** = `true` 🔗

- void **set_editable**(value: bool)
- bool **is_editable**()

If `true`, the **SpinBox** will be editable. Otherwise, it will be read only.

String **prefix** = `""` 🔗

- void **set_prefix**(value: String)
- String **get_prefix**()

Adds the specified prefix string before the numerical value of the **SpinBox**.

---

bool **select_all_on_focus** = `false` 🔗

- void **set_select_all_on_focus**(value: bool)
- bool **is_select_all_on_focus**()

If `true`, the **SpinBox** will select the whole text when the LineEdit gains focus. Clicking the up and down arrows won't trigger this behavior.

String **suffix** = `""` 🔗

- void **set_suffix**(value: String)
- String **get_suffix**()

Adds the specified suffix string after the numerical value of the **SpinBox**.

---

bool **update_on_text_changed** = `false` 🔗

- void **set_update_on_text_changed**(value: bool)
- bool **get_update_on_text_changed**()

Sets the value of the Range for this **SpinBox** when the LineEdit text is *changed* instead of *submitted*. See LineEdit.text_changed and LineEdit.text_submitted.

Method Descriptions

void **apply**() 🔗

Applies the current value of this **SpinBox**.

LineEdit **get_line_edit**() 🔗

Returns the LineEdit instance from this **SpinBox**. You can use it to access properties and methods of LineEdit.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their CanvasItem.visible property.

Theme Property Descriptions

Texture2D **updown**

Sets a custom Texture2D for up and down arrows of the **SpinBox**.

SplitContainer

Inherits: Container **<** Control **<** CanvasItem **<** Node **<** Object

Inherited By: HSplitContainer, VSplitContainer

A container that splits two child controls horizontally or vertically and provides a grabber for adjusting the split ratio.

Description

A container that accepts only two child controls, then arranges them horizontally or vertically and creates a divisor between them. The divisor can be dragged around to change the size relation between the child controls.

Tutorials

- Using Containers

Properties

bool	collapsed	`false`
DraggerVisibility	dragger_visibility	`0`
int	split_offset	`0`

bool	vertical	`false`

Methods

void	clamp_split_offset()

Theme Properties

int	autohide	`1`
int	minimum_grab_thickness	`6`
int	separation	`12`
Texture2D	grabber	
Texture2D	h_grabber	
Texture2D	v_grabber	

Signals

dragged(offset: int) 🔗

Emitted when the dragger is dragged by user.

Enumerations

enum **DraggerVisibility**: 🔗

DraggerVisibility **DRAGGER_VISIBLE** = `0`

The split dragger is visible when the cursor hovers it.

DraggerVisibility **DRAGGER_HIDDEN** = `1`

The split dragger is never visible.

DraggerVisibility **DRAGGER_HIDDEN_COLLAPSED** = `2`

The split dragger is never visible and its space collapsed.

Property Descriptions

bool **collapsed** = `false` 🔗

- void **set_collapsed**(value: bool)
- bool **is_collapsed**()

If `true`, the area of the first Control will be collapsed and the dragger will be disabled.

DraggerVisibility **dragger_visibility** = `0` 🔗

- void **set_dragger_visibility**(value: DraggerVisibility)
- DraggerVisibility **get_dragger_visibility**()

Determines the dragger's visibility. See DraggerVisibility for details.

int **split_offset** = `0` 🔗

- void **set_split_offset**(value: int)
- int **get_split_offset**()

The initial offset of the splitting between the two Controls, with `0` being at the end of the first Control.

bool **vertical** = `false` 🔗

- void **set_vertical**(value: bool)
- bool **is_vertical**()

If `true`, the **SplitContainer** will arrange its children vertically, rather than horizontally.

Can't be changed when using HSplitContainer and VSplitContainer.

Method Descriptions

void **clamp_split_offset**() 🔗

Clamps the split_offset value to not go outside the currently possible minimal and maximum values.

Theme Property Descriptions

int **autohide** = `1`

Boolean value. If 1 (`true`), the grabber will hide automatically when it isn't under the cursor. If 0 (`false`), it's always visible.

int **minimum_grab_thickness** = `6`

The minimum thickness of the area users can click on to grab the splitting line. If separation or h_grabber / v_grabber's thickness are too small, this ensure that the splitting line can still be dragged.

int **separation** = `12`

The space between sides of the container.

Texture2D **grabber**

The icon used for the grabber drawn in the middle area.

Texture2D **h_grabber**

The icon used for the grabber drawn in the middle area when vertical is `false`.

Texture2D **v_grabber**

The icon used for the grabber drawn in the middle area when vertical is `true`.

SpotLight3D

Inherits: Light3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A spotlight, such as a reflector spotlight or a lantern.

Description

A Spotlight is a type of Light3D node that emits lights in a specific direction, in the shape of a cone. The light is attenuated through the distance. This attenuation can be configured by changing the energy, radius and attenuation parameters of Light3D.

Note: When using the Mobile rendering method, only 8 spot lights can be displayed on each mesh resource. Attempting to display more than 8 spot lights on a single mesh resource will result in spot lights flickering in and out as the camera moves. When using the Compatibility rendering method, only 8 spot lights can be displayed on each mesh resource by default, but this can be increased by adjusting ProjectSettings.rendering/limits/opengl/max_lights_per_object.

Note: When using the Mobile or Compatibility rendering methods, spot lights will only correctly affect meshes whose visibility AABB intersects with the light's AABB. If using a shader to deform the mesh in a way that makes it go outside its AABB, GeometryInstance3D.extra_cull_margin must be increased on the mesh. Otherwise, the light may not be visible on the mesh.

Tutorials

- 3D lights and shadows

- Faking global illumination
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

float	shadow_bias	`0.03` (overrides Light3D)
float	shadow_normal_bias	`1.0` (overrides Light3D)
float	spot_angle	`45.0`
float	spot_angle_attenuation	`1.0`
float	spot_attenuation	`1.0`
float	spot_range	`5.0`

Property Descriptions

float **spot_angle** = `45.0` 🔗

- void **set_param**(value: float)
- float **get_param**()

The spotlight's angle in degrees.

Note: spot_angle is not affected by Node3D.scale (the light's scale or its parent's scale).

---

float **spot_angle_attenuation** = `1.0` 🔗

- void **set_param**(value: float)
- float **get_param**()

The spotlight's *angular* attenuation curve. See also spot_attenuation.

---

float **spot_attenuation** = `1.0` 🔗

- void **set_param**(value: float)
- float **get_param**()

Controls the distance attenuation function for spotlights.

A value of `0.0` will maintain a constant brightness through most of the range, but smoothly attenuate the light at the edge of the range. Use a value of `2.0` for physically accurate lights as it results in the proper inverse square attenutation.

Note: Setting attenuation to `2.0` or higher may result in distant objects receiving minimal light, even within range. For example, with a range of `4096`, an object at `100` units is attenuated by a factor of `0.0001`. With a default brightness of `1`, the light would not be visible at that distance.

Note: Using negative or values higher than `10.0` may lead to unexpected results.

---

float **spot_range** = `5.0` 🔗

- void **set_param**(value: float)

- float **get_param**()

The maximal range that can be reached by the spotlight. Note that the effectively lit area may appear to be smaller depending on the spot_attenuation in use. No matter the spot_attenuation in use, the light will never reach anything outside this range.

Note: spot_range is not affected by Node3D.scale (the light's scale or its parent's scale).

SpringArm3D

Inherits: Node3D **<** Node **<** Object

A 3D raycast that dynamically moves its children near the collision point.

Description

SpringArm3D casts a ray or a shape along its Z axis and moves all its direct children to the collision point, with an optional margin. This is useful for 3rd person cameras that move closer to the player when inside a tight space (you may need to exclude the player's collider from the **SpringArm3D**'s collision check).

Properties

int	collision_mask	`1`
float	margin	`0.01`
Shape3D	shape	
float	spring_length	`1.0`

Methods

void	add_excluded_object(RID: RID)
void	clear_excluded_objects()
float	get_hit_length()
bool	remove_excluded_object(RID: RID)

Property Descriptions

int **collision_mask** = `1` 🔗

- void **set_collision_mask**(value: int)
- int **get_collision_mask**()

The layers against which the collision check shall be done. See Collision layers and masks in the documentation for more information.

---

float **margin** = `0.01` 🔗

- void **set_margin**(value: float)
- float **get_margin**()

When the collision check is made, a candidate length for the SpringArm3D is given.

The margin is then subtracted to this length and the translation is applied to the child objects of the SpringArm3D.

This margin is useful for when the SpringArm3D has a Camera3D as a child node: without the margin, the Camera3D would be placed on the exact point of collision, while with the margin the Camera3D would be placed close to the point of collision.

Shape3D **shape** 🔗

- void **set_shape**(value: Shape3D)
- Shape3D **get_shape**()

The Shape3D to use for the SpringArm3D.

When the shape is set, the SpringArm3D will cast the Shape3D on its z axis instead of performing a ray cast.

---

float **spring_length** = `1.0` 🔗

- void **set_length**(value: float)
- float **get_length**()

The maximum extent of the SpringArm3D. This is used as a length for both the ray and the shape cast used internally to calculate the desired position of the SpringArm3D's child nodes.

To know more about how to perform a shape cast or a ray cast, please consult the PhysicsDirectSpaceState3D documentation.

Method Descriptions

void **add_excluded_object**(RID: RID) 🔗

Adds the PhysicsBody3D object with the given RID to the list of PhysicsBody3D objects excluded from the collision check.

void **clear_excluded_objects**()

Clears the list of PhysicsBody3D objects excluded from the collision check.

float **get_hit_length**()

Returns the spring arm's current length.

bool **remove_excluded_object**(RID: RID)

Removes the given RID from the list of PhysicsBody3D objects excluded from the collision check.

Sprite2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

General-purpose sprite node.

Description

A node that displays a 2D texture. The texture displayed can be a region from a larger atlas texture, or a frame from a sprite sheet animation.

Tutorials

- Instancing Demo [https://godotengine.org/asset-library/asset/2716]

Properties

bool	centered	`true`
bool	flip_h	`false`
bool	flip_v	`false`
int	frame	`0`

Vector2i	frame_coords	`Vector2i(0, 0)`
int	hframes	`1`
Vector2	offset	`Vector2(0, 0)`
bool	region_enabled	`false`
bool	region_filter_clip_enabled	`false`
Rect2	region_rect	`Rect2(0, 0, 0, 0)`
Texture2D	texture	
int	vframes	`1`

Methods

Rect2	get_rect() const
bool	is_pixel_opaque(pos: Vector2) const

Signals

frame_changed() 🔗

Emitted when the frame changes.

---

texture_changed() 🔗

Emitted when the texture changes.

Property Descriptions

bool **centered** = `true` 🔗

- void **set_centered**(value: bool)
- bool **is_centered**()

If `true`, texture is centered.

Note: For games with a pixel art aesthetic, textures may appear deformed when centered. This is caused by their position being between pixels. To prevent this, set this property to `false`, or consider enabling ProjectSettings.rendering/2d/snap/snap_2d_vertices_to_pixel and ProjectSettings.rendering/2d/snap/snap_2d_transforms_to_pixel.

---

bool **flip_h** = `false` 🔗

- void **set_flip_h**(value: bool)
- bool **is_flipped_h**()

If `true`, texture is flipped horizontally.

---

bool **flip_v** = `false` 🔗

- void **set_flip_v**(value: bool)
- bool **is_flipped_v**()

If `true`, texture is flipped vertically.

---

int **frame** = `0`

- void **set_frame**(value: int)
- int **get_frame**()

Current frame to display from sprite sheet. hframes or vframes must be greater than 1. This property is automatically adjusted when hframes or vframes are changed to keep pointing to the same visual frame (same column and row). If that's impossible, this value is reset to `0`.

Vector2i **frame_coords** = `Vector2i(0, 0)`

- void **set_frame_coords**(value: Vector2i)
- Vector2i **get_frame_coords**()

Coordinates of the frame to display from sprite sheet. This is as an alias for the frame property. hframes or vframes must be greater than 1.

---

int **hframes** = `1`

- void **set_hframes**(value: int)
- int **get_hframes**()

The number of columns in the sprite sheet. When this property is changed, frame is adjusted so that the same visual frame is maintained (same row and column). If that's impossible, frame is reset to `0`.

Vector2 **offset** = `Vector2(0, 0)`

- void **set_offset**(value: Vector2)
- Vector2 **get_offset**()

The texture's drawing offset.

bool **region_enabled** = `false`

- void **set_region_enabled**(value: bool)
- bool **is_region_enabled**()

If `true`, texture is cut from a larger atlas texture. See region_rect.

bool **region_filter_clip_enabled** = `false`

- void **set_region_filter_clip_enabled**(value: bool)
- bool **is_region_filter_clip_enabled**()

If `true`, the outermost pixels get blurred out. region_enabled must be `true`.

Rect2 **region_rect** = `Rect2(0, 0, 0, 0)`

- void **set_region_rect**(value: Rect2)
- Rect2 **get_region_rect**()

The region of the atlas texture to display. region_enabled must be `true`.

Texture2D **texture**

- void **set_texture**(value: Texture2D)
- Texture2D **get_texture**()

Texture2D object to draw.

---

int **vframes** = `1`

- void **set_vframes**(value: int)
- int **get_vframes**()

The number of rows in the sprite sheet. When this property is changed, frame is adjusted so that the same visual frame is maintained (same row and column). If that's impossible, frame is reset to `0`.

Method Descriptions

Rect2 **get_rect**() const

Returns a Rect2 representing the Sprite2D's boundary in local coordinates. Can be used to detect if the Sprite2D was clicked.

Example:

GDScript

```
func _input(event):
    if event is InputEventMouseButton and event.pressed and
event.button_index == MOUSE_BUTTON_LEFT:
        if get_rect().has_point(to_local(event.position)):
            print("A click!")
```

C#

```
public override void _Input(InputEvent @event)
{
    if (@event is InputEventMouseButton inputEventMouse)
    {
        if (inputEventMouse.Pressed && inputEventMouse.ButtonIndex == MouseButton.Left)
        {
            if (GetRect().HasPoint(ToLocal(inputEventMouse.Position)))
            {
                GD.Print("A click!");
            }
        }
    }
}
```

---

bool **is_pixel_opaque**(pos: Vector2) const

Returns `true`, if the pixel at the given position is opaque and `false` in other case. The position is in local coordinates.

Note: It also returns `false`, if the sprite's texture is `null` or if the given position is invalid.

Sprite3D

Inherits: SpriteBase3D **<** GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

2D sprite node in a 3D world.

Description

A node that displays a 2D texture in a 3D environment. The texture displayed can be a region from a larger atlas texture, or a frame from a sprite sheet animation. See also SpriteBase3D where properties such as the billboard mode are defined.

Properties

int	frame	`0`
Vector2i	frame_coords	`Vector2i(0, 0)`
int	hframes	`1`
bool	region_enabled	`false`
Rect2	region_rect	`Rect2(0, 0, 0, 0)`

Texture2D	texture	
int	vframes	`1`

Signals

frame_changed() 🔗

Emitted when the frame changes.

---

texture_changed() 🔗

Emitted when the texture changes.

Property Descriptions

int **frame** = `0` 🔗

- void **set_frame**(value: int)
- int **get_frame**()

Current frame to display from sprite sheet. hframes or vframes must be greater than 1. This property is automatically adjusted when hframes or vframes are changed to keep pointing to the same visual frame (same column and row). If that's impossible, this value is reset to `0`.

Vector2i **frame_coords** = `Vector2i(0, 0)` 🔗

- void **set_frame_coords**(value: Vector2i)
- Vector2i **get_frame_coords**()

Coordinates of the frame to display from sprite sheet. This is as an alias for the frame property. hframes or vframes must be greater than 1.

---

int **hframes** = `1`

- void **set_hframes**(value: int)
- int **get_hframes**()

The number of columns in the sprite sheet. When this property is changed, frame is adjusted so that the same visual frame is maintained (same row and column). If that's impossible, frame is reset to `0`.

---

bool **region_enabled** = `false`

- void **set_region_enabled**(value: bool)
- bool **is_region_enabled**()

If `true`, the sprite will use region_rect and display only the specified part of its texture.

Rect2 **region_rect** = `Rect2(0, 0, 0, 0)`

- void **set_region_rect**(value: Rect2)
- Rect2 **get_region_rect**()

The region of the atlas texture to display. region_enabled must be `true`.

Texture2D **texture**

- void **set_texture**(value: Texture2D)
- Texture2D **get_texture**()

Texture2D object to draw. If GeometryInstance3D.material_override is used, this will be overridden. The size information is still used.

---

int **vframes** = `1`

- void **set_vframes**(value: int)
- int **get_vframes**()

The number of rows in the sprite sheet. When this property is changed, frame is adjusted so that the same visual frame is maintained (same row and column). If that's impossible, frame is reset to `0`.

SpriteBase3D

Inherits: GeometryInstance3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

Inherited By: AnimatedSprite3D, Sprite3D

2D sprite node in 3D environment.

Description

A node that displays 2D texture information in a 3D environment. See also Sprite3D where many other properties are defined.

Properties

float	alpha_antialiasing_edge	`0.0`
AlphaAntiAliasing	alpha_antialiasing_mode	`0`
AlphaCutMode	alpha_cut	`0`
float	alpha_hash_scale	`1.0`
float	alpha_scissor_threshold	`0.5`
Vector3.Axis	axis	`2`

BillboardMode	billboard	`0`
bool	centered	`true`
bool	double_sided	`true`
bool	fixed_size	`false`
bool	flip_h	`false`
bool	flip_v	`false`
Color	modulate	`Color(1, 1, 1, 1)`
bool	no_depth_test	`false`
Vector2	offset	`Vector2(0, 0)`
float	pixel_size	`0.01`
int	render_priority	`0`
bool	shaded	`false`

TextureFilter	texture_filter	`3`
bool	transparent	`true`

Methods

TriangleMesh	generate_triangle_mesh() const
bool	get_draw_flag(flag: DrawFlags) const
Rect2	get_item_rect() const
void	set_draw_flag(flag: DrawFlags, enabled: bool)

Enumerations

enum **DrawFlags**: 🔗

DrawFlags **FLAG_TRANSPARENT** = `0`

If set, the texture's transparency and the opacity are used to make those parts of the sprite invisible.

DrawFlags **FLAG_SHADED** = `1`

If set, lights in the environment affect the sprite.

DrawFlags **FLAG_DOUBLE_SIDED** = `2`

If set, texture can be seen from the back as well. If not, the texture is invisible when looking at it from behind.

DrawFlags **FLAG_DISABLE_DEPTH_TEST** = `3`

Disables the depth test, so this object is drawn on top of all others. However, objects drawn after it in the draw order may cover it.

DrawFlags **FLAG_FIXED_SIZE** = `4`

Label is scaled by depth so that it always appears the same size on screen.

DrawFlags **FLAG_MAX** = `5`

Represents the size of the DrawFlags enum.

enum **AlphaCutMode**: 🔗

AlphaCutMode **ALPHA_CUT_DISABLED** = `0`

This mode performs standard alpha blending. It can display translucent areas, but transparency sorting issues may be visible when multiple transparent materials are overlapping.

AlphaCutMode **ALPHA_CUT_DISCARD** = `1`

This mode only allows fully transparent or fully opaque pixels. Harsh edges will be visible unless some form of screen-space antialiasing is enabled (see ProjectSettings.rendering/anti_aliasing/quality/screen_space_aa). On the bright side, this mode doesn't suffer from transparency sorting issues when multiple transparent materials are overlapping. This mode is also known as *alpha testing* or *1-bit transparency*.

AlphaCutMode **ALPHA_CUT_OPAQUE_PREPASS** = `2`

This mode draws fully opaque pixels in the depth prepass. This is slower than ALPHA_CUT_DISABLED or ALPHA_CUT_DISCARD, but it allows displaying translucent areas and smooth edges while using proper sorting.

AlphaCutMode **ALPHA_CUT_HASH** = `3`

This mode draws cuts off all values below a spatially-deterministic threshold, the rest will remain opaque.

Property Descriptions

float **alpha_antialiasing_edge** = `0.0`

- void **set_alpha_antialiasing_edge**(value: float)
- float **get_alpha_antialiasing_edge**()

Threshold at which antialiasing will be applied on the alpha channel.

AlphaAntiAliasing **alpha_antialiasing_mode** = `0`

- void **set_alpha_antialiasing**(value: AlphaAntiAliasing)
- AlphaAntiAliasing **get_alpha_antialiasing**()

The type of alpha antialiasing to apply. See AlphaAntiAliasing.

AlphaCutMode **alpha_cut** = `0`

- void **set_alpha_cut_mode**(value: AlphaCutMode)
- AlphaCutMode **get_alpha_cut_mode**()

The alpha cutting mode to use for the sprite. See AlphaCutMode for possible values.

---

float **alpha_hash_scale** = `1.0`

- void **set_alpha_hash_scale**(value: float)
- float **get_alpha_hash_scale**()

The hashing scale for Alpha Hash. Recommended values between `0` and `2`.

---

float **alpha_scissor_threshold** = `0.5`

- void **set_alpha_scissor_threshold**(value: float)
- float **get_alpha_scissor_threshold**()

Threshold at which the alpha scissor will discard values.

Vector3.Axis **axis** = `2`

- void **set_axis**(value: Vector3.Axis)
- Vector3.Axis **get_axis**()

The direction in which the front of the texture faces.

BillboardMode **billboard** = `0`

- void **set_billboard_mode**(value: BillboardMode)
- BillboardMode **get_billboard_mode**()

The billboard mode to use for the sprite. See BillboardMode for possible values.

Note: When billboarding is enabled and the material also casts shadows, billboards will face **the** camera in the scene when rendering shadows. In scenes with multiple cameras, the intended shadow cannot be determined and this will result in undefined behavior. See GitHub Pull Request #72638 [https://github.com/godotengine/godot/pull/72638] for details.

---

bool **centered** = `true` 🔗

- void **set_centered**(value: bool)
- bool **is_centered**()

If `true`, texture will be centered.

---

bool **double_sided** = `true` 🔗

- void **set_draw_flag**(flag: DrawFlags, enabled: bool)
- bool **get_draw_flag**(flag: DrawFlags) const

If `true`, texture can be seen from the back as well, if `false`, it is invisible when looking at it from behind.

---

bool **fixed_size** = `false` 🔗

- void **set_draw_flag**(flag: DrawFlags, enabled: bool)
- bool **get_draw_flag**(flag: DrawFlags) const

If `true`, the label is rendered at the same size regardless of distance.

---

bool **flip_h** = `false` 🔗

- void **set_flip_h**(value: bool)
- bool **is_flipped_h**()

If `true`, texture is flipped horizontally.

---

bool **flip_v** = `false`

- void **set_flip_v**(value: bool)
- bool **is_flipped_v**()

If `true`, texture is flipped vertically.

Color **modulate** = `Color(1, 1, 1, 1)`

- void **set_modulate**(value: Color)
- Color **get_modulate**()

A color value used to *multiply* the texture's colors. Can be used for mood-coloring or to simulate the color of ambient light.

Note: Unlike CanvasItem.modulate for 2D, colors with values above `1.0` (overbright) are not supported.

Note: If a GeometryInstance3D.material_override is defined on the **SpriteBase3D**, the material override must be configured to take vertex colors into account for albedo. Otherwise, the color defined in modulate will be ignored. For a BaseMaterial3D, BaseMaterial3D.vertex_color_use_as_albedo must be `true`. For a ShaderMaterial, `ALBEDO *= COLOR.rgb;` must be inserted in the shader's `fragment()` function.

---

bool **no_depth_test** = `false`

- void **set_draw_flag**(flag: DrawFlags, enabled: bool)
- bool **get_draw_flag**(flag: DrawFlags) const

If `true`, depth testing is disabled and the object will be drawn in render order.

Vector2 **offset** = `Vector2(0, 0)`

- void **set_offset**(value: Vector2)
- Vector2 **get_offset**()

The texture's drawing offset.

---

float **pixel_size** = `0.01`

- void **set_pixel_size**(value: float)
- float **get_pixel_size**()

The size of one pixel's width on the sprite to scale it in 3D.

---

int **render_priority** = `0`

- void **set_render_priority**(value: int)
- int **get_render_priority**()

Sets the render priority for the sprite. Higher priority objects will be sorted in front of lower priority objects.

Note: This only applies if alpha_cut is set to ALPHA_CUT_DISABLED (default value).

Note: This only applies to sorting of transparent objects. This will not impact how transparent objects are sorted relative to opaque objects. This is because opaque objects are not sorted, while transparent objects are sorted from back to front (subject to priority).

---

bool **shaded** = `false` 🔗

- void **set_draw_flag**(flag: DrawFlags, enabled: bool)
- bool **get_draw_flag**(flag: DrawFlags) const

If `true`, the Light3D in the Environment has effects on the sprite.

TextureFilter **texture_filter** = `3` 🔗

- void **set_texture_filter**(value: TextureFilter)
- TextureFilter **get_texture_filter**()

Filter flags for the texture. See TextureFilter for options.

Note: Linear filtering may cause artifacts around the edges, which are especially noticeable on opaque textures. To prevent this, use textures with transparent or identical colors around the edges.

---

bool **transparent** = `true` 🔗

- void **set_draw_flag**(flag: DrawFlags, enabled: bool)
- bool **get_draw_flag**(flag: DrawFlags) const

If `true`, the texture's transparency and the opacity are used to make those parts of the sprite invisible.

Method Descriptions

TriangleMesh **generate_triangle_mesh**() const 🔗

Returns a TriangleMesh with the sprite's vertices following its current configuration (such as its axis and pixel_size).

bool **get_draw_flag**(flag: DrawFlags) const 🔗

Returns the value of the specified flag.

Rect2 **get_item_rect**() const 🔗

Returns the rectangle representing this sprite.

---

void **set_draw_flag**(flag: DrawFlags, enabled: bool) 🔗

If `true`, the specified flag will be enabled. See DrawFlags for a list of flags.

StaticBody2D

Inherits: PhysicsBody2D **<** CollisionObject2D **<** Node2D **<** CanvasItem **<** Node **<** Object

Inherited By: AnimatableBody2D

A 2D physics body that can't be moved by external forces. When moved manually, it doesn't affect other bodies in its path.

Description

A static 2D physics body. It can't be moved by external forces or contacts, but can be moved manually by other means such as code, AnimationMixers (with AnimationMixer.callback_mode_process set to AnimationMixer.ANIMATION_CALLBACK_MODE_PROCESS_PHYSICS), and RemoteTransform2D.

When **StaticBody2D** is moved, it is teleported to its new position without affecting other physics bodies in its path. If this is not desired, use AnimatableBody2D instead.

StaticBody2D is useful for completely static objects like floors and walls, as well as moving surfaces like conveyor belts and circular revolving platforms (by using constant_linear_velocity and constant_angular_velocity).

Properties

float	constant_angular_velocity	`0.0`

Vector2	constant_linear_velocity	`Vector2(0, 0)`
PhysicsMaterial	physics_material_override	

Property Descriptions

float **constant_angular_velocity** = `0.0`

- void **set_constant_angular_velocity**(value: float)
- float **get_constant_angular_velocity**()

The body's constant angular velocity. This does not rotate the body, but affects touching bodies, as if it were rotating.

Vector2 **constant_linear_velocity** = `Vector2(0, 0)`

- void **set_constant_linear_velocity**(value: Vector2)
- Vector2 **get_constant_linear_velocity**()

The body's constant linear velocity. This does not move the body, but affects touching bodies, as if it were moving.

PhysicsMaterial **physics_material_override**

- void **set_physics_material_override**(value: PhysicsMaterial)
- PhysicsMaterial **get_physics_material_override**()

The physics material override for the body.

If a material is assigned to this property, it will be used instead of any other physics material, such as an inherited one.

StaticBody3D

Inherits: PhysicsBody3D **<** CollisionObject3D **<** Node3D **<** Node **<** Object

Inherited By: AnimatableBody3D

A 3D physics body that can't be moved by external forces. When moved manually, it doesn't affect other bodies in its path.

Description

A static 3D physics body. It can't be moved by external forces or contacts, but can be moved manually by other means such as code, AnimationMixers (with AnimationMixer.callback_mode_process set to AnimationMixer.ANIMATION_CALLBACK_MODE_PROCESS_PHYSICS), and RemoteTransform3D.

When **StaticBody3D** is moved, it is teleported to its new position without affecting other physics bodies in its path. If this is not desired, use AnimatableBody3D instead.

StaticBody3D is useful for completely static objects like floors and walls, as well as moving surfaces like conveyor belts and circular revolving platforms (by using constant_linear_velocity and constant_angular_velocity).

Tutorials

- 3D Physics Tests Demo [https://godotengine.org/asset-library/asset/2747]

- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]
- 3D Voxel Demo [https://godotengine.org/asset-library/asset/2755]

Properties

Vector3	constant_angular_velocity	`Vector3(0, 0, 0)`
Vector3	constant_linear_velocity	`Vector3(0, 0, 0)`
PhysicsMaterial	physics_material_override	

Property Descriptions

Vector3 **constant_angular_velocity** = `Vector3(0, 0, 0)` 🔗

- void **set_constant_angular_velocity**(value: Vector3)
- Vector3 **get_constant_angular_velocity**()

The body's constant angular velocity. This does not rotate the body, but affects touching bodies, as if it were rotating.

Vector3 **constant_linear_velocity** = `Vector3(0, 0, 0)` 🔗

- void **set_constant_linear_velocity**(value: Vector3)
- Vector3 **get_constant_linear_velocity**()

The body's constant linear velocity. This does not move the body, but affects touching bodies, as if it were moving.

PhysicsMaterial **physics_material_override**

- void **set_physics_material_override**(value: PhysicsMaterial)
- PhysicsMaterial **get_physics_material_override**()

The physics material override for the body.

If a material is assigned to this property, it will be used instead of any other physics material, such as an inherited one.

StatusIndicator

Inherits: Node < Object

Application status indicator (aka notification area icon).

Note: Status indicator is implemented on macOS and Windows.

Properties

Texture2D	icon	
NodePath	menu	`NodePath("")`
String	tooltip	`""`
bool	visible	`true`

Methods

Rect2	get_rect() const

Signals

pressed(mouse_button: int, mouse_position: Vector2i) 🔗

Emitted when the status indicator is pressed.

Property Descriptions

Texture2D **icon** 🔗

- void **set_icon**(value: Texture2D)
- Texture2D **get_icon**()

Status indicator icon.

NodePath **menu** = `NodePath("")` 🔗

- void **set_menu**(value: NodePath)
- NodePath **get_menu**()

Status indicator native popup menu. If this is set, the pressed signal is not emitted.

Note: Native popup is only supported if NativeMenu supports NativeMenu.FEATURE_POPUP_MENU feature.

String **tooltip** = `""` 🔗

- void **set_tooltip**(value: String)
- String **get_tooltip**()

Status indicator tooltip.

---

bool **visible** = `true` 🔗

- void **set_visible**(value: bool)
- bool **is_visible**()

If `true`, the status indicator is visible.

Method Descriptions

Rect2 **get_rect**() const

Returns the status indicator rectangle in screen coordinates. If this status indicator is not visible, returns an empty Rect2.

SubViewport

Inherits: Viewport **<** Node **<** Object

An interface to a game world that doesn't create a window or draw to the screen directly.

Description

SubViewport Isolates a rectangular region of a scene to be displayed independently. This can be used, for example, to display UI in 3D space.

Note: **SubViewport** is a Viewport that isn't a Window, i.e. it doesn't draw anything by itself. To display anything, **SubViewport** must have a non-zero size and be either put inside a SubViewportContainer or assigned to a ViewportTexture.

Tutorials

- Using Viewports
- Viewport and canvas transforms
- GUI in 3D Viewport Demo [https://godotengine.org/asset-library/asset/2807]
- 3D in 2D Viewport Demo [https://godotengine.org/asset-library/asset/2804]
- 2D in 3D Viewport Demo [https://godotengine.org/asset-library/asset/2803]
- Screen Capture Demo [https://godotengine.org/asset-library/asset/2808]
- Dynamic Split Screen Demo [https://godotengine.org/asset-library/asset/2806]

- 3D Resolution Scaling Demo [https://godotengine.org/asset-library/asset/2805]

Properties

ClearMode	render_target_clear_mode	`0`
UpdateMode	render_target_update_mode	`2`
Vector2i	size	`Vector2i(512, 512)`
Vector2i	size_2d_override	`Vector2i(0, 0)`
bool	size_2d_override_stretch	`false`

Enumerations

enum **ClearMode**: 🔗

ClearMode **CLEAR_MODE_ALWAYS** = `0`

Always clear the render target before drawing.

ClearMode **CLEAR_MODE_NEVER** = `1`

Never clear the render target.

ClearMode **CLEAR_MODE_ONCE** = `2`

Clear the render target on the next frame, then switch to CLEAR_MODE_NEVER.

enum **UpdateMode**: 🔗

UpdateMode **UPDATE_DISABLED** = `0`

Do not update the render target.

UpdateMode **UPDATE_ONCE** = `1`

Update the render target once, then switch to UPDATE_DISABLED.

UpdateMode **UPDATE_WHEN_VISIBLE** = `2`

Update the render target only when it is visible. This is the default value.

UpdateMode **UPDATE_WHEN_PARENT_VISIBLE** = `3`

Update the render target only when its parent is visible.

UpdateMode **UPDATE_ALWAYS** = `4`

Always update the render target.

Property Descriptions

ClearMode **render_target_clear_mode** = `0` 🔗

- void **set_clear_mode**(value: ClearMode)
- ClearMode **get_clear_mode**()

The clear mode when the sub-viewport is used as a render target.

Note: This property is intended for 2D usage.

UpdateMode **render_target_update_mode** = `2`

- void **set_update_mode**(value: UpdateMode)
- UpdateMode **get_update_mode**()

The update mode when the sub-viewport is used as a render target.

Vector2i **size** = `Vector2i(512, 512)`

- void **set_size**(value: Vector2i)
- Vector2i **get_size**()

The width and height of the sub-viewport. Must be set to a value greater than or equal to 2 pixels on both dimensions. Otherwise, nothing will be displayed.

Note: If the parent node is a SubViewportContainer and its SubViewportContainer.stretch is `true`, the viewport size cannot be changed manually.

Vector2i **size_2d_override** = `Vector2i(0, 0)`

- void **set_size_2d_override**(value: Vector2i)
- Vector2i **get_size_2d_override**()

The 2D size override of the sub-viewport. If either the width or height is `0`, the override is disabled.

---

bool **size_2d_override_stretch** = `false`

- void **set_size_2d_override_stretch**(value: bool)

- bool **is_size_2d_override_stretch_enabled**()

If `true`, the 2D size override affects stretch as well.

SubViewportContainer

Inherits: Container **<** Control **<** CanvasItem **<** Node **<** Object

A container used for displaying the contents of a SubViewport.

Description

A container that displays the contents of underlying SubViewport child nodes. It uses the combined size of the SubViewports as minimum size, unless stretch is enabled.

Note: Changing a **SubViewportContainer**'s Control.scale will cause its contents to appear distorted. To change its visual size without causing distortion, adjust the node's margins instead (if it's not already in a container).

Note: The **SubViewportContainer** forwards mouse-enter and mouse-exit notifications to its sub-viewports.

Properties

FocusMode	focus_mode	`1` (overrides Control)
bool	stretch	`false`
int	stretch_shrink	`1`

Methods

bool	**_propagate_input_event**(event: InputEvent) virtual const

---

Property Descriptions

bool **stretch** = `false` 🔗

- void **set_stretch**(value: bool)
- bool **is_stretch_enabled**()

If `true`, the sub-viewport will be automatically resized to the control's size.

Note: If `true`, this will prohibit changing SubViewport.size of its children manually.

---

int **stretch_shrink** = `1` 🔗

- void **set_stretch_shrink**(value: int)
- int **get_stretch_shrink**()

Divides the sub-viewport's effective resolution by this value while preserving its scale. This can be used to speed up rendering.

For example, a 1280×720 sub-viewport with stretch_shrink set to `2` will be rendered at 640×360 while occupying the same size in the container.

Note: stretch must be `true` for this property to work.

Method Descriptions

bool **_propagate_input_event**(event: InputEvent) virtual const

Experimental: This method may be changed or removed in future versions.

Virtual method to be implemented by the user. If it returns `true`, the `event` is propagated to SubViewport children. Propagation doesn't happen if it returns `false`. If the function is not implemented, all events are propagated to SubViewports.

TabBar

Inherits: Control **<** CanvasItem **<** Node **<** Object

A control that provides a horizontal bar with tabs.

Description

A control that provides a horizontal bar with tabs. Similar to TabContainer but is only in charge of drawing tabs, not interacting with children.

Properties

bool	clip_tabs	`true`
int	current_tab	`-1`
bool	deselect_enabled	`false`
bool	drag_to_rearrange_enabled	`false`
FocusMode	focus_mode	`2` (overrides Control)
int	max_tab_width	`0`

bool	scroll_to_selected	`true`
bool	scrolling_enabled	`true`
bool	select_with_rmb	`false`
AlignmentMode	tab_alignment	`0`
CloseButtonDisplayPolicy	tab_close_display_policy	`0`
int	tab_count	`0`
int	tabs_rearrange_group	`-1`

Methods

void	add_tab(title: String = "", icon: Texture2D = null)
void	clear_tabs()
void	ensure_tab_visible(idx: int)
bool	get_offset_buttons_visible() const

int	get_previous_tab() const
Texture2D	get_tab_button_icon(tab_idx: int) const
Texture2D	get_tab_icon(tab_idx: int) const
int	get_tab_icon_max_width(tab_idx: int) const
int	get_tab_idx_at_point(point: Vector2) const
String	get_tab_language(tab_idx: int) const
Variant	get_tab_metadata(tab_idx: int) const
int	get_tab_offset() const
Rect2	get_tab_rect(tab_idx: int) const
TextDirection	get_tab_text_direction(tab_idx: int) const
String	get_tab_title(tab_idx: int) const
String	get_tab_tooltip(tab_idx: int) const
bool	is_tab_disabled(tab_idx: int) const

bool	is_tab_hidden(tab_idx: int) const
void	move_tab(from: int, to: int)
void	remove_tab(tab_idx: int)
bool	select_next_available()
bool	select_previous_available()
void	set_tab_button_icon(tab_idx: int, icon: Texture2D)
void	set_tab_disabled(tab_idx: int, disabled: bool)
void	set_tab_hidden(tab_idx: int, hidden: bool)
void	set_tab_icon(tab_idx: int, icon: Texture2D)
void	set_tab_icon_max_width(tab_idx: int, width: int)
void	set_tab_language(tab_idx: int, language: String)
void	set_tab_metadata(tab_idx: int, metadata: Variant)

void	set_tab_text_direction(tab_idx: int, direction: TextDirection)
void	set_tab_title(tab_idx: int, title: String)
void	set_tab_tooltip(tab_idx: int, tooltip: String)

Theme Properties

Color	drop_mark_color	`Color(1, 1, 1, 1)`
Color	font_disabled_color	`Color(0.875, 0.875, 0.875, 0.5)`
Color	font_hovered_color	`Color(0.95, 0.95, 0.95, 1)`
Color	font_outline_color	`Color(0, 0, 0, 1)`
Color	font_selected_color	`Color(0.95, 0.95, 0.95, 1)`
Color	font_unselected_color	`Color(0.7, 0.7, 0.7, 1)`
int	h_separation	`4`

int	icon_max_width	0
int	outline_size	0
Font	font	
int	font_size	
Texture2D	close	
Texture2D	decrement	
Texture2D	decrement_highlight	
Texture2D	drop_mark	
Texture2D	increment	
Texture2D	increment_highlight	
StyleBox	button_highlight	
StyleBox	button_pressed	
StyleBox	tab_disabled	

StyleBox	tab_focus
StyleBox	tab_hovered
StyleBox	tab_selected
StyleBox	tab_unselected

Signals

active_tab_rearranged(idx_to: int)

Emitted when the active tab is rearranged via mouse drag. See drag_to_rearrange_enabled.

tab_button_pressed(tab: int)

Emitted when a tab's right button is pressed. See set_tab_button_icon.

tab_changed(tab: int)

Emitted when switching to another tab.

tab_clicked(tab: int)

Emitted when a tab is clicked, even if it is the current tab.

---

tab_close_pressed(tab: int)

Emitted when a tab's close button is pressed.

Note: Tabs are not removed automatically once the close button is pressed, this behavior needs to be programmed manually. For example:

GDScript

```
$TabBar.tab_close_pressed.connect($TabBar.remove_tab)
```

C#

```
GetNode<TabBar>("TabBar").TabClosePressed += GetNode<TabBar>
("TabBar").RemoveTab;
```

---

tab_hovered(tab: int)

Emitted when a tab is hovered by the mouse.

---

tab_rmb_clicked(tab: int)

Emitted when a tab is right-clicked. select_with_rmb must be enabled.

---

tab_selected(tab: int)

Emitted when a tab is selected via click, directional input, or script, even if it is the current tab.

Enumerations

enum **AlignmentMode**: 🔗

AlignmentMode **ALIGNMENT_LEFT** = `0`

Places tabs to the left.

AlignmentMode **ALIGNMENT_CENTER** = `1`

Places tabs in the middle.

AlignmentMode **ALIGNMENT_RIGHT** = `2`

Places tabs to the right.

AlignmentMode **ALIGNMENT_MAX** = `3`

Represents the size of the AlignmentMode enum.

enum **CloseButtonDisplayPolicy**: 🔗

CloseButtonDisplayPolicy **CLOSE_BUTTON_SHOW_NEVER** = `0`

Never show the close buttons.

CloseButtonDisplayPolicy **CLOSE_BUTTON_SHOW_ACTIVE_ONLY** = `1`

Only show the close button on the currently active tab.

CloseButtonDisplayPolicy **CLOSE_BUTTON_SHOW_ALWAYS** = `2`

Show the close button on all tabs.

CloseButtonDisplayPolicy **CLOSE_BUTTON_MAX** = `3`

Represents the size of the CloseButtonDisplayPolicy enum.

Property Descriptions

bool **clip_tabs** = `true` 🔗

- void **set_clip_tabs**(value: bool)
- bool **get_clip_tabs**()

If `true`, tabs overflowing this node's width will be hidden, displaying two navigation buttons instead. Otherwise, this node's minimum size is updated so that all tabs are visible.

---

int **current_tab** = `-1` 🔗

- void **set_current_tab**(value: int)
- int **get_current_tab**()

The index of the current selected tab. A value of `-1` means that no tab is selected and can only be set when deselect_enabled is `true` or if all tabs are hidden or disabled.

---

bool **deselect_enabled** = `false` 🔗

- void **set_deselect_enabled**(value: bool)
- bool **get_deselect_enabled**()

If `true`, all tabs can be deselected so that no tab is selected. Click on the current tab to deselect it.

---

bool **drag_to_rearrange_enabled** = `false` 🔗

- void **set_drag_to_rearrange_enabled**(value: bool)
- bool **get_drag_to_rearrange_enabled**()

If `true`, tabs can be rearranged with mouse drag.

---

int **max_tab_width** = `0`

- void **set_max_tab_width**(value: int)
- int **get_max_tab_width**()

Sets the maximum width which all tabs should be limited to. Unlimited if set to `0`.

---

bool **scroll_to_selected** = `true`

- void **set_scroll_to_selected**(value: bool)
- bool **get_scroll_to_selected**()

If `true`, the tab offset will be changed to keep the currently selected tab visible.

---

bool **scrolling_enabled** = `true`

- void **set_scrolling_enabled**(value: bool)
- bool **get_scrolling_enabled**()

if `true`, the mouse's scroll wheel can be used to navigate the scroll view.

---

bool **select_with_rmb** = `false`

- void **set_select_with_rmb**(value: bool)
- bool **get_select_with_rmb**()

If `true`, enables selecting a tab with the right mouse button.

AlignmentMode **tab_alignment** = `0`

- void **set_tab_alignment**(value: AlignmentMode)
- AlignmentMode **get_tab_alignment**()

Sets the position at which tabs will be placed. See AlignmentMode for details.

CloseButtonDisplayPolicy **tab_close_display_policy** = `0`

- void **set_tab_close_display_policy**(value: CloseButtonDisplayPolicy)
- CloseButtonDisplayPolicy **get_tab_close_display_policy**()

Sets when the close button will appear on the tabs. See CloseButtonDisplayPolicy for details.

int **tab_count** = `0`

- void **set_tab_count**(value: int)
- int **get_tab_count**()

The number of tabs currently in the bar.

int **tabs_rearrange_group** = `-1`

- void **set_tabs_rearrange_group**(value: int)
- int **get_tabs_rearrange_group**()

TabBars with the same rearrange group ID will allow dragging the tabs between them. Enable drag with drag_to_rearrange_enabled.

Setting this to `-1` will disable rearranging between **TabBar**s.

Method Descriptions

void **add_tab**(title: String = "", icon: Texture2D = null) 🔗

Adds a new tab.

---

void **clear_tabs**() 🔗

Clears all tabs.

---

void **ensure_tab_visible**(idx: int) 🔗

Moves the scroll view to make the tab visible.

---

bool **get_offset_buttons_visible**() const 🔗

Returns `true` if the offset buttons (the ones that appear when there's not enough space for all tabs) are visible.

---

int **get_previous_tab**() const 🔗

Returns the previously active tab index.

Texture2D **get_tab_button_icon**(tab_idx: int) const 🔗

Returns the icon for the right button of the tab at index `tab_idx` or `null` if the right button has no icon.

Texture2D **get_tab_icon**(tab_idx: int) const 🔗

Returns the icon for the tab at index `tab_idx` or `null` if the tab has no icon.

int **get_tab_icon_max_width**(tab_idx: int) const 🔗

Returns the maximum allowed width of the icon for the tab at index `tab_idx`.

int **get_tab_idx_at_point**(point: Vector2) const 🔗

Returns the index of the tab at local coordinates `point`. Returns `-1` if the point is outside the control boundaries or if there's no tab at the queried position.

String **get_tab_language**(tab_idx: int) const 🔗

Returns tab title language code.

Variant **get_tab_metadata**(tab_idx: int) const 🔗

Returns the metadata value set to the tab at index `tab_idx` using set_tab_metadata. If no metadata was previously set, returns `null` by default.

int **get_tab_offset**() const 🔗

Returns the number of hidden tabs offsetted to the left.

Rect2 **get_tab_rect**(tab_idx: int) const 🔗

Returns tab Rect2 with local position and size.

TextDirection **get_tab_text_direction**(tab_idx: int) const

Returns tab title text base writing direction.

String **get_tab_title**(tab_idx: int) const

Returns the title of the tab at index `tab_idx`.

String **get_tab_tooltip**(tab_idx: int) const

Returns the tooltip text of the tab at index `tab_idx`.

bool **is_tab_disabled**(tab_idx: int) const

Returns `true` if the tab at index `tab_idx` is disabled.

bool **is_tab_hidden**(tab_idx: int) const

Returns `true` if the tab at index `tab_idx` is hidden.

void **move_tab**(from: int, to: int)

Moves a tab from `from` to `to`.

void **remove_tab**(tab_idx: int)

Removes the tab at index `tab_idx`.

bool **select_next_available**()

Selects the first available tab with greater index than the currently selected. Returns `true` if tab selection changed.

---

bool **select_previous_available**() 🔗

Selects the first available tab with lower index than the currently selected. Returns `true` if tab selection changed.

---

void **set_tab_button_icon**(tab_idx: int, icon: Texture2D) 🔗

Sets an `icon` for the button of the tab at index `tab_idx` (located to the right, before the close button), making it visible and clickable (See tab_button_pressed). Giving it a `null` value will hide the button.

---

void **set_tab_disabled**(tab_idx: int, disabled: bool) 🔗

If `disabled` is `true`, disables the tab at index `tab_idx`, making it non-interactable.

---

void **set_tab_hidden**(tab_idx: int, hidden: bool) 🔗

If `hidden` is `true`, hides the tab at index `tab_idx`, making it disappear from the tab area.

---

void **set_tab_icon**(tab_idx: int, icon: Texture2D) 🔗

Sets an `icon` for the tab at index `tab_idx`.

---

void **set_tab_icon_max_width**(tab_idx: int, width: int) 🔗

Sets the maximum allowed width of the icon for the tab at index `tab_idx`. This limit is applied on top of the default size of the icon and on top of icon_max_width. The height is adjusted according to the icon's ratio.

---

void **set_tab_language**(tab_idx: int, language: String)

Sets language code of tab title used for line-breaking and text shaping algorithms, if left empty current locale is used instead.

---

void **set_tab_metadata**(tab_idx: int, metadata: Variant)

Sets the metadata value for the tab at index `tab_idx`, which can be retrieved later using get_tab_metadata.

---

void **set_tab_text_direction**(tab_idx: int, direction: TextDirection)

Sets tab title base writing direction.

---

void **set_tab_title**(tab_idx: int, title: String)

Sets a `title` for the tab at index `tab_idx`.

---

void **set_tab_tooltip**(tab_idx: int, tooltip: String)

Sets a `tooltip` for tab at index `tab_idx`.

Note: By default, if the `tooltip` is empty and the tab text is truncated (not all characters fit into the tab), the title will be displayed as a tooltip. To hide the tooltip, assign `" "` as the `tooltip` text.

Theme Property Descriptions

Color **drop_mark_color** = `Color(1, 1, 1, 1)`

Modulation color for the drop_mark icon.

Color **font_disabled_color** = `Color(0.875, 0.875, 0.875, 0.5)`

Font color of disabled tabs.

Color **font_hovered_color** = `Color(0.95, 0.95, 0.95, 1)`

Font color of the currently hovered tab. Does not apply to the selected tab.

Color **font_outline_color** = `Color(0, 0, 0, 1)`

The tint of text outline of the tab name.

Color **font_selected_color** = `Color(0.95, 0.95, 0.95, 1)`

Font color of the currently selected tab.

Color **font_unselected_color** = `Color(0.7, 0.7, 0.7, 1)`

Font color of the other, unselected tabs.

---

int **h_separation** = `4` 🔗

The horizontal separation between the elements inside tabs.

---

int **icon_max_width** = `0` 🔗

The maximum allowed width of the tab's icon. This limit is applied on top of the default size of the icon, but before the value set with set_tab_icon_max_width. The height is adjusted according to the icon's ratio.

---

int **outline_size** = `0` 🔗

The size of the tab text outline.

Note: If using a font with FontFile.multichannel_signed_distance_field enabled, its FontFile.msdf_pixel_range must be set to at least *twice* the value of outline_size for outline rendering to look correct. Otherwise, the outline may appear to be cut off earlier than intended.

Font **font** 🔗

The font used to draw tab names.

---

int **font_size** 🔗

Font size of the tab names.

Texture2D **close** 🔗

The icon for the close button (see tab_close_display_policy).

Texture2D **decrement** 🔗

Icon for the left arrow button that appears when there are too many tabs to fit in the container width. When the button is disabled (i.e. the first tab is visible), it appears semi-transparent.

Texture2D **decrement_highlight** 🔗

Icon for the left arrow button that appears when there are too many tabs to fit in the container width. Used when the button is being hovered with the cursor.

Texture2D **drop_mark** 🔗

Icon shown to indicate where a dragged tab is gonna be dropped (see drag_to_rearrange_enabled).

Texture2D **increment** 🔗

Icon for the right arrow button that appears when there are too many tabs to fit in the container width. When the button is disabled (i.e. the last tab is visible) it appears semi-transparent.

Texture2D **increment_highlight** 🔗

Icon for the right arrow button that appears when there are too many tabs to fit in the container width. Used when the button is being hovered with the cursor.

StyleBox **button_highlight** 🔗

Background of the tab and close buttons when they're being hovered with the cursor.

StyleBox **button_pressed** 🔗

Background of the tab and close buttons when it's being pressed.

StyleBox **tab_disabled** 🔗

The style of disabled tabs.

StyleBox **tab_focus** 🔗

StyleBox used when the **TabBar** is focused. The tab_focus StyleBox is displayed *over* the base StyleBox of the selected tab, so a partially transparent StyleBox should be used to ensure the base StyleBox remains visible. A StyleBox that represents an outline or an underline works well for this purpose. To disable the focus visual effect, assign a StyleBoxEmpty resource. Note that disabling the focus visual effect will harm keyboard/controller navigation usability, so this is not recommended for accessibility reasons.

StyleBox **tab_hovered** 🔗

The style of the currently hovered tab. Does not apply to the selected tab.

Note: This style will be drawn with the same width as tab_unselected at minimum.

StyleBox **tab_selected**

The style of the currently selected tab.

StyleBox **tab_unselected**

The style of the other, unselected tabs.

TabContainer

Inherits: Container **<** Control **<** CanvasItem **<** Node **<** Object

A container that creates a tab for each child control, displaying only the active tab's control.

Description

Arranges child controls into a tabbed view, creating a tab for each one. The active tab's corresponding control is made visible, while all other child controls are hidden. Ignores non-control children.

Note: The drawing of the clickable tabs is handled by this node; TabBar is not needed.

Tutorials

- Using Containers

Properties

bool	all_tabs_in_front	`false`
bool	clip_tabs	`true`
int	current_tab	`-1`

bool	deselect_enabled	`false`
bool	drag_to_rearrange_enabled	`false`
AlignmentMode	tab_alignment	`0`
FocusMode	tab_focus_mode	`2`
TabPosition	tabs_position	`0`
int	tabs_rearrange_group	`-1`
bool	tabs_visible	`true`
bool	use_hidden_tabs_for_min_size	`false`

Methods

Control	get_current_tab_control() const
Popup	get_popup() const
int	get_previous_tab() const

TabBar	get_tab_bar() const
Texture2D	get_tab_button_icon(tab_idx: int) const
Control	get_tab_control(tab_idx: int) const
int	get_tab_count() const
Texture2D	get_tab_icon(tab_idx: int) const
int	get_tab_icon_max_width(tab_idx: int) const
int	get_tab_idx_at_point(point: Vector2) const
int	get_tab_idx_from_control(control: Control) const
Variant	get_tab_metadata(tab_idx: int) const
String	get_tab_title(tab_idx: int) const
String	get_tab_tooltip(tab_idx: int) const
bool	is_tab_disabled(tab_idx: int) const
bool	is_tab_hidden(tab_idx: int) const

bool	select_next_available()
bool	select_previous_available()
void	set_popup(popup: Node)
void	set_tab_button_icon(tab_idx: int, icon: Texture2D)
void	set_tab_disabled(tab_idx: int, disabled: bool)
void	set_tab_hidden(tab_idx: int, hidden: bool)
void	set_tab_icon(tab_idx: int, icon: Texture2D)
void	set_tab_icon_max_width(tab_idx: int, width: int)
void	set_tab_metadata(tab_idx: int, metadata: Variant)
void	set_tab_title(tab_idx: int, title: String)
void	set_tab_tooltip(tab_idx: int, tooltip: String)

Theme Properties

Color	drop_mark_color	`Color(1, 1, 1, 1)`
Color	font_disabled_color	`Color(0.875, 0.875, 0.875, 0.5)`
Color	font_hovered_color	`Color(0.95, 0.95, 0.95, 1)`
Color	font_outline_color	`Color(0, 0, 0, 1)`
Color	font_selected_color	`Color(0.95, 0.95, 0.95, 1)`
Color	font_unselected_color	`Color(0.7, 0.7, 0.7, 1)`
int	icon_max_width	`0`
int	icon_separation	`4`
int	outline_size	`0`
int	side_margin	`8`
Font	font	
int	font_size	

Texture2D	decrement
Texture2D	decrement_highlight
Texture2D	drop_mark
Texture2D	increment
Texture2D	increment_highlight
Texture2D	menu
Texture2D	menu_highlight
StyleBox	panel
StyleBox	tab_disabled
StyleBox	tab_focus
StyleBox	tab_hovered
StyleBox	tab_selected
StyleBox	tab_unselected

StyleBox	tabbar_background

Signals

active_tab_rearranged(idx_to: int) 🔗

Emitted when the active tab is rearranged via mouse drag. See drag_to_rearrange_enabled.

pre_popup_pressed() 🔗

Emitted when the **TabContainer**'s Popup button is clicked. See set_popup for details.

tab_button_pressed(tab: int) 🔗

Emitted when the user clicks on the button icon on this tab.

tab_changed(tab: int) 🔗

Emitted when switching to another tab.

tab_clicked(tab: int) 🔗

Emitted when a tab is clicked, even if it is the current tab.

tab_hovered(tab: int) 🔗

Emitted when a tab is hovered by the mouse.

---

tab_selected(tab: int)

Emitted when a tab is selected via click, directional input, or script, even if it is the current tab.

Enumerations

enum **TabPosition**:

TabPosition **POSITION_TOP** = `0`

Places the tab bar at the top.

TabPosition **POSITION_BOTTOM** = `1`

Places the tab bar at the bottom. The tab bar's StyleBox will be flipped vertically.

TabPosition **POSITION_MAX** = `2`

Represents the size of the TabPosition enum.

Property Descriptions

bool **all_tabs_in_front** = `false`

- void **set_all_tabs_in_front**(value: bool)
- bool **is_all_tabs_in_front**()

If `true`, all tabs are drawn in front of the panel. If `false`, inactive tabs are drawn behind the panel.

---

bool **clip_tabs** = `true`

- void **set_clip_tabs**(value: bool)
- bool **get_clip_tabs**()

If `true`, tabs overflowing this node's width will be hidden, displaying two navigation buttons instead. Otherwise, this node's minimum size is updated so that all tabs are visible.

---

int **current_tab** = `-1`

- void **set_current_tab**(value: int)
- int **get_current_tab**()

The current tab index. When set, this index's Control node's `visible` property is set to `true` and all others are set to `false`.

A value of `-1` means that no tab is selected.

---

bool **deselect_enabled** = `false`

- void **set_deselect_enabled**(value: bool)
- bool **get_deselect_enabled**()

If `true`, all tabs can be deselected so that no tab is selected. Click on the current_tab to deselect it.

Only the tab header will be shown if no tabs are selected.

---

bool **drag_to_rearrange_enabled** = `false`

- void **set_drag_to_rearrange_enabled**(value: bool)
- bool **get_drag_to_rearrange_enabled**()

If `true`, tabs can be rearranged with mouse drag.

AlignmentMode **tab_alignment** = `0` 🔗

- void **set_tab_alignment**(value: AlignmentMode)
- AlignmentMode **get_tab_alignment**()

Sets the position at which tabs will be placed. See AlignmentMode for details.

FocusMode **tab_focus_mode** = `2` 🔗

- void **set_tab_focus_mode**(value: FocusMode)
- FocusMode **get_tab_focus_mode**()

The focus access mode for the internal TabBar node.

TabPosition **tabs_position** = `0` 🔗

- void **set_tabs_position**(value: TabPosition)
- TabPosition **get_tabs_position**()

Sets the position of the tab bar. See TabPosition for details.

---

int **tabs_rearrange_group** = `-1` 🔗

- void **set_tabs_rearrange_group**(value: int)
- int **get_tabs_rearrange_group**()

TabContainers with the same rearrange group ID will allow dragging the tabs between them. Enable drag with drag_to_rearrange_enabled.

Setting this to `-1` will disable rearranging between **TabContainer**s.

---

bool **tabs_visible** = `true` 🔗

- void **set_tabs_visible**(value: bool)
- bool **are_tabs_visible**()

If `true`, tabs are visible. If `false`, tabs' content and titles are hidden.

---

bool **use_hidden_tabs_for_min_size** = `false` 🔗

- void **set_use_hidden_tabs_for_min_size**(value: bool)
- bool **get_use_hidden_tabs_for_min_size**()

If `true`, child Control nodes that are hidden have their minimum size take into account in the total, instead of only the currently visible one.

Method Descriptions

Control **get_current_tab_control**() const 🔗

Returns the child Control node located at the active tab index.

Popup **get_popup**() const 🔗

Returns the Popup node instance if one has been set already with set_popup.

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their Window.visible property.

---

int **get_previous_tab**() const

Returns the previously active tab index.

TabBar **get_tab_bar**() const

Returns the TabBar contained in this container.

Warning: This is a required internal node, removing and freeing it or editing its tabs may cause a crash. If you wish to edit the tabs, use the methods provided in **TabContainer**.

Texture2D **get_tab_button_icon**(tab_idx: int) const

Returns the button icon from the tab at index `tab_idx`.

Control **get_tab_control**(tab_idx: int) const

Returns the Control node from the tab at index `tab_idx`.

---

int **get_tab_count**() const

Returns the number of tabs.

Texture2D **get_tab_icon**(tab_idx: int) const

Returns the Texture2D for the tab at index `tab_idx` or `null` if the tab has no Texture2D.

---

int **get_tab_icon_max_width**(tab_idx: int) const

Returns the maximum allowed width of the icon for the tab at index `tab_idx`.

---

int **get_tab_idx_at_point**(point: Vector2) const

Returns the index of the tab at local coordinates `point`. Returns `-1` if the point is outside the control boundaries or if there's no tab at the queried position.

---

int **get_tab_idx_from_control**(control: Control) const

Returns the index of the tab tied to the given `control`. The control must be a child of the **TabContainer**.

Variant **get_tab_metadata**(tab_idx: int) const

Returns the metadata value set to the tab at index `tab_idx` using set_tab_metadata. If no metadata was previously set, returns `null` by default.

String **get_tab_title**(tab_idx: int) const

Returns the title of the tab at index `tab_idx`. Tab titles default to the name of the indexed child node, but this can be overridden with set_tab_title.

String **get_tab_tooltip**(tab_idx: int) const 🔗

Returns the tooltip text of the tab at index `tab_idx`.

---

bool **is_tab_disabled**(tab_idx: int) const 🔗

Returns `true` if the tab at index `tab_idx` is disabled.

---

bool **is_tab_hidden**(tab_idx: int) const 🔗

Returns `true` if the tab at index `tab_idx` is hidden.

---

bool **select_next_available**() 🔗

Selects the first available tab with greater index than the currently selected. Returns `true` if tab selection changed.

---

bool **select_previous_available**() 🔗

Selects the first available tab with lower index than the currently selected. Returns `true` if tab selection changed.

---

void **set_popup**(popup: Node) 🔗

If set on a Popup node instance, a popup menu icon appears in the top-right corner of the **TabContainer** (setting it to `null` will make it go away). Clicking it will expand the Popup node.

---

void **set_tab_button_icon**(tab_idx: int, icon: Texture2D) 🔗

Sets the button icon from the tab at index `tab_idx`.

void **set_tab_disabled**(tab_idx: int, disabled: bool)

If `disabled` is `true`, disables the tab at index `tab_idx`, making it non-interactable.

void **set_tab_hidden**(tab_idx: int, hidden: bool)

If `hidden` is `true`, hides the tab at index `tab_idx`, making it disappear from the tab area.

void **set_tab_icon**(tab_idx: int, icon: Texture2D)

Sets an icon for the tab at index `tab_idx`.

void **set_tab_icon_max_width**(tab_idx: int, width: int)

Sets the maximum allowed width of the icon for the tab at index `tab_idx`. This limit is applied on top of the default size of the icon and on top of icon_max_width. The height is adjusted according to the icon's ratio.

void **set_tab_metadata**(tab_idx: int, metadata: Variant)

Sets the metadata value for the tab at index `tab_idx`, which can be retrieved later using get_tab_metadata.

void **set_tab_title**(tab_idx: int, title: String)

Sets a custom title for the tab at index `tab_idx` (tab titles default to the name of the indexed child node). Set it back to the child's name to make the tab default to it again.

---

void **set_tab_tooltip**(tab_idx: int, tooltip: String)

Sets a custom tooltip text for tab at index `tab_idx`.

Note: By default, if the `tooltip` is empty and the tab text is truncated (not all characters fit into the tab), the title will be displayed as a tooltip. To hide the tooltip, assign `" "` as the `tooltip` text.

Theme Property Descriptions

Color **drop_mark_color** = `Color(1, 1, 1, 1)`

Modulation color for the drop_mark icon.

Color **font_disabled_color** = `Color(0.875, 0.875, 0.875, 0.5)`

Font color of disabled tabs.

Color **font_hovered_color** = `Color(0.95, 0.95, 0.95, 1)`

Font color of the currently hovered tab.

Color **font_outline_color** = `Color(0, 0, 0, 1)`

The tint of text outline of the tab name.

Color **font_selected_color** = `Color(0.95, 0.95, 0.95, 1)` 🔗

Font color of the currently selected tab.

Color **font_unselected_color** = `Color(0.7, 0.7, 0.7, 1)` 🔗

Font color of the other, unselected tabs.

int **icon_max_width** = `0` 🔗

The maximum allowed width of the tab's icon. This limit is applied on top of the default size of the icon, but before the value set with TabBar.set_tab_icon_max_width. The height is adjusted according to the icon's ratio.

int **icon_separation** = `4` 🔗

Space between tab's name and its icon.

int **outline_size** = `0` 🔗

The size of the tab text outline.

Note: If using a font with FontFile.multichannel_signed_distance_field enabled, its FontFile.msdf_pixel_range must be set to at least *twice* the value of outline_size for outline rendering to look correct. Otherwise, the outline may appear to be cut off earlier than intended.

int **side_margin** = `8` 🔗

The space at the left or right edges of the tab bar, accordingly with the current tab_alignment.

The margin is ignored with TabBar.ALIGNMENT_RIGHT if the tabs are clipped (see clip_tabs) or a popup has been set (see set_popup). The margin is always ignored with TabBar.ALIGNMENT_CENTER.

Font **font** 🔗

The font used to draw tab names.

int **font_size** 🔗

Font size of the tab names.

Texture2D **decrement** 🔗

Icon for the left arrow button that appears when there are too many tabs to fit in the container width. When the button is disabled (i.e. the first tab is visible), it appears semi-transparent.

Texture2D **decrement_highlight** 🔗

Icon for the left arrow button that appears when there are too many tabs to fit in the container width. Used when the button is being hovered with the cursor.

Texture2D **drop_mark** 🔗

Icon shown to indicate where a dragged tab is gonna be dropped (see drag_to_rearrange_enabled).

Texture2D **increment**

Icon for the right arrow button that appears when there are too many tabs to fit in the container width. When the button is disabled (i.e. the last tab is visible) it appears semi-transparent.

Texture2D **increment_highlight**

Icon for the right arrow button that appears when there are too many tabs to fit in the container width. Used when the button is being hovered with the cursor.

Texture2D **menu**

The icon for the menu button (see set_popup).

Texture2D **menu_highlight**

The icon for the menu button (see set_popup) when it's being hovered with the cursor.

StyleBox **panel**

The style for the background fill.

StyleBox **tab_disabled**

The style of disabled tabs.

StyleBox **tab_focus**

StyleBox used when the TabBar is focused. The tab_focus StyleBox is displayed *over* the base StyleBox of the selected tab, so a partially transparent StyleBox should be used to ensure the base StyleBox remains visible. A StyleBox that represents an outline or an underline works well for this purpose. To disable the focus visual effect, assign a StyleBoxEmpty resource. Note that disabling the focus visual effect will harm keyboard/controller navigation usability, so this is not recommended for accessibility reasons.

StyleBox **tab_hovered**

The style of the currently hovered tab.

Note: This style will be drawn with the same width as tab_unselected at minimum.

StyleBox **tab_selected**

The style of the currently selected tab.

StyleBox **tab_unselected**

The style of the other, unselected tabs.

StyleBox **tabbar_background**

The style for the background fill of the TabBar area.

TextEdit

Inherits: Control **<** CanvasItem **<** Node **<** Object

Inherited By: CodeEdit

A multiline text editor.

Description

A multiline text editor. It also has limited facilities for editing code, such as syntax highlighting support. For more advanced facilities for editing code, see CodeEdit.

Note: Most viewport, caret, and edit methods contain a `caret_index` argument for caret_multiple support. The argument should be one of the following: `-1` for all carets, `0` for the main caret, or greater than `0` for secondary carets in the order they were created.

Note: When holding down `Alt`, the vertical scroll wheel will scroll 5 times as fast as it would normally do. This also works in the Godot script editor.

Properties

AutowrapMode	autowrap_mode	`3`
bool	caret_blink	`false`
float	caret_blink_interval	`0.65`

bool	caret_draw_when_editable_disabled	`false`
bool	caret_mid_grapheme	`false`
bool	caret_move_on_right_click	`true`
bool	caret_multiple	`true`
CaretType	caret_type	`0`
bool	clip_contents	`true` (overrides Control)
bool	context_menu_enabled	`true`
String	custom_word_separators	`""`
bool	deselect_on_focus_loss_enabled	`true`
bool	drag_and_drop_selection_enabled	`true`
bool	draw_control_chars	`false`
bool	draw_spaces	`false`

bool	draw_tabs	`false`
bool	editable	`true`
FocusMode	focus_mode	`2` (overrides Control)
bool	highlight_all_occurrences	`false`
bool	highlight_current_line	`false`
bool	indent_wrapped_lines	`false`
String	language	`""`
bool	middle_mouse_paste_enabled	`true`
bool	minimap_draw	`false`
int	minimap_width	`80`
CursorShape	mouse_default_cursor_shape	`1` (overrides Control)
String	placeholder_text	`""`

bool	scroll_fit_content_height	`false`
int	scroll_horizontal	`0`
bool	scroll_past_end_of_file	`false`
bool	scroll_smooth	`false`
float	scroll_v_scroll_speed	`80.0`
float	scroll_vertical	`0.0`
bool	selecting_enabled	`true`
bool	shortcut_keys_enabled	`true`
StructuredTextParser	structured_text_bidi_override	`0`
Array	structured_text_bidi_override_options	`[]`
SyntaxHighlighter	syntax_highlighter	
String	text	`""`
TextDirection	text_direction	`0`

bool	use_custom_word_separators	`false`
bool	use_default_word_separators	`true`
bool	virtual_keyboard_enabled	`true`
LineWrappingMode	wrap_mode	`0`

Methods

void	_backspace(caret_index: int) virtual
void	_copy(caret_index: int) virtual
void	_cut(caret_index: int) virtual
void	_handle_unicode_input(unicode_char: int, caret_index: int) virtual
void	_paste(caret_index: int) virtual
void	_paste_primary_clipboard(caret_index: int) virtual
int	add_caret(line: int, column: int)

void	add_caret_at_carets(below: bool)
void	add_gutter(at: int = -1)
void	add_selection_for_next_occurrence()
void	adjust_carets_after_edit(caret: int, from_line: int, from_col: int, to_line: int, to_col: int)
void	adjust_viewport_to_caret(caret_index: int = 0)
void	apply_ime()
void	backspace(caret_index: int = -1)
void	begin_complex_operation()
void	begin_multicaret_edit()
void	cancel_ime()
void	center_viewport_to_caret(caret_index: int = 0)
void	clear()
void	clear_undo_history()

void	collapse_carets(from_line: int, from_column: int, to_line: int, to_column: int, inclusive: bool = false)
void	copy(caret_index: int = -1)
void	cut(caret_index: int = -1)
void	delete_selection(caret_index: int = -1)
void	deselect(caret_index: int = -1)
void	end_action()
void	end_complex_operation()
void	end_multicaret_edit()
int	get_caret_column(caret_index: int = 0) const
int	get_caret_count() const
Vector2	get_caret_draw_pos(caret_index: int = 0) const
PackedInt32Array	get_caret_index_edit_order()
int	get_caret_line(caret_index: int = 0) const

int	get_caret_wrap_index(caret_index: int = 0) const
int	get_first_non_whitespace_column(line: int) const
int	get_first_visible_line() const
int	get_gutter_count() const
String	get_gutter_name(gutter: int) const
GutterType	get_gutter_type(gutter: int) const
int	get_gutter_width(gutter: int) const
HScrollBar	get_h_scroll_bar() const
int	get_indent_level(line: int) const
int	get_last_full_visible_line() const
int	get_last_full_visible_line_wrap_index() const
int	get_last_unhidden_line() const
String	get_line(line: int) const

Color	get_line_background_color(line: int) const
Vector2i	get_line_column_at_pos(position: Vector2i, allow_out_of_bounds: bool = true) const
int	get_line_count() const
Texture2D	get_line_gutter_icon(line: int, gutter: int) const
Color	get_line_gutter_item_color(line: int, gutter: int) const
Variant	get_line_gutter_metadata(line: int, gutter: int) const
String	get_line_gutter_text(line: int, gutter: int) const
int	get_line_height() const
Array[Vector2i]	get_line_ranges_from_carets(only_selections: bool = false, merge_adjacent: bool = true) const
int	get_line_width(line: int, wrap_index: int = -1) const
int	get_line_wrap_count(line: int) const

int	get_line_wrap_index_at_column(line: int, column: int) const
PackedStringArray	get_line_wrapped_text(line: int) const
Vector2	get_local_mouse_pos() const
PopupMenu	get_menu() const
int	get_minimap_line_at_pos(position: Vector2i) const
int	get_minimap_visible_lines() const
Vector2i	get_next_visible_line_index_offset_from(line: int, wrap_index: int, visible_amount: int) const
int	get_next_visible_line_offset_from(line: int, visible_amount: int) const
Vector2i	get_pos_at_line_column(line: int, column: int) const
Rect2i	get_rect_at_line_column(line: int, column: int) const
int	get_saved_version() const

float	get_scroll_pos_for_line(line: int, wrap_index: int = 0) const
String	get_selected_text(caret_index: int = -1)
int	get_selection_at_line_column(line: int, column: int, include_edges: bool = true, only_selections: bool = true) const
int	get_selection_column(caret_index: int = 0) const
int	get_selection_from_column(caret_index: int = 0) const
int	get_selection_from_line(caret_index: int = 0) const
int	get_selection_line(caret_index: int = 0) const
SelectionMode	get_selection_mode() const
int	get_selection_origin_column(caret_index: int = 0) const
int	get_selection_origin_line(caret_index: int = 0) const

int	get_selection_to_column(caret_index: int = 0) const
int	get_selection_to_line(caret_index: int = 0) const
PackedInt32Array	get_sorted_carets(include_ignored_carets: bool = false) const
int	get_tab_size() const
int	get_total_gutter_width() const
int	get_total_visible_line_count() const
VScrollBar	get_v_scroll_bar() const
int	get_version() const
int	get_visible_line_count() const
int	get_visible_line_count_in_range(from_line: int, to_line: int) const
String	get_word_at_pos(position: Vector2) const
String	get_word_under_caret(caret_index: int = -1) const

bool	has_ime_text() const
bool	has_redo() const
bool	has_selection(caret_index: int = -1) const
bool	has_undo() const
void	insert_line_at(line: int, text: String)
void	insert_text(text: String, line: int, column: int, before_selection_begin: bool = true, before_selection_end: bool = false)
void	insert_text_at_caret(text: String, caret_index: int = -1)
bool	is_caret_after_selection_origin(caret_index: int = 0) const
bool	is_caret_visible(caret_index: int = 0) const
bool	is_dragging_cursor() const
bool	is_gutter_clickable(gutter: int) const
bool	is_gutter_drawn(gutter: int) const

bool	is_gutter_overwritable(gutter: int) const
bool	is_in_mulitcaret_edit() const
bool	is_line_gutter_clickable(line: int, gutter: int) const
bool	is_line_wrapped(line: int) const
bool	is_menu_visible() const
bool	is_mouse_over_selection(edges: bool, caret_index: int = -1) const
bool	is_overtype_mode_enabled() const
void	menu_option(option: int)
void	merge_gutters(from_line: int, to_line: int)
void	merge_overlapping_carets()
bool	multicaret_edit_ignore_caret(caret_index: int) const
void	paste(caret_index: int = -1)
void	paste_primary_clipboard(caret_index: int = -1)

void	redo()
void	remove_caret(caret: int)
void	remove_gutter(gutter: int)
void	remove_line_at(line: int, move_carets_down: bool = true)
void	remove_secondary_carets()
void	remove_text(from_line: int, from_column: int, to_line: int, to_column: int)
Vector2i	search(text: String, flags: int, from_line: int, from_column: int) const
void	select(origin_line: int, origin_column: int, caret_line: int, caret_column: int, caret_index: int = 0)
void	select_all()
void	select_word_under_caret(caret_index: int = -1)
void	set_caret_column(column: int, adjust_viewport: bool = true, caret_index: int = 0)

void	set_caret_line(line: int, adjust_viewport: bool = true, can_be_hidden: bool = true, wrap_index: int = 0, caret_index: int = 0)
void	set_gutter_clickable(gutter: int, clickable: bool)
void	set_gutter_custom_draw(column: int, draw_callback: Callable)
void	set_gutter_draw(gutter: int, draw: bool)
void	set_gutter_name(gutter: int, name: String)
void	set_gutter_overwritable(gutter: int, overwritable: bool)
void	set_gutter_type(gutter: int, type: GutterType)
void	set_gutter_width(gutter: int, width: int)
void	set_line(line: int, new_text: String)
void	set_line_as_center_visible(line: int, wrap_index: int = 0)
void	set_line_as_first_visible(line: int, wrap_index: int = 0)

void	set_line_as_last_visible(line: int, wrap_index: int = 0)
void	set_line_background_color(line: int, color: Color)
void	set_line_gutter_clickable(line: int, gutter: int, clickable: bool)
void	set_line_gutter_icon(line: int, gutter: int, icon: Texture2D)
void	set_line_gutter_item_color(line: int, gutter: int, color: Color)
void	set_line_gutter_metadata(line: int, gutter: int, metadata: Variant)
void	set_line_gutter_text(line: int, gutter: int, text: String)
void	set_overtype_mode_enabled(enabled: bool)
void	set_search_flags(flags: int)
void	set_search_text(search_text: String)
void	set_selection_mode(mode: SelectionMode)

void	set_selection_origin_column(column: int, caret_index: int = 0)
void	set_selection_origin_line(line: int, can_be_hidden: bool = true, wrap_index: int = -1, caret_index: int = 0)
void	set_tab_size(size: int)
void	set_tooltip_request_func(callback: Callable)
void	skip_selection_for_next_occurrence()
void	start_action(action: EditAction)
void	swap_lines(from_line: int, to_line: int)
void	tag_saved_version()
void	undo()

Theme Properties

Color	background_color	`Color(0, 0, 0, 0)`
Color	caret_background_color	`Color(0, 0, 0, 1)`

Color	caret_color	`Color(0.875, 0.875, 0.875, 1)`
Color	current_line_color	`Color(0.25, 0.25, 0.26, 0.8)`
Color	font_color	`Color(0.875, 0.875, 0.875, 1)`
Color	font_outline_color	`Color(0, 0, 0, 1)`
Color	font_placeholder_color	`Color(0.875, 0.875, 0.875, 0.6)`
Color	font_readonly_color	`Color(0.875, 0.875, 0.875, 0.5)`
Color	font_selected_color	`Color(0, 0, 0, 0)`
Color	search_result_border_color	`Color(0.3, 0.3, 0.3, 0.4)`
Color	search_result_color	`Color(0.3, 0.3, 0.3, 1)`
Color	selection_color	`Color(0.5, 0.5, 0.5, 1)`
Color	word_highlighted_color	`Color(0.5, 0.5, 0.5, 0.25)`
int	caret_width	`1`

int	line_spacing	`4`
int	outline_size	`0`
Font	font	
int	font_size	
Texture2D	space	
Texture2D	tab	
StyleBox	focus	
StyleBox	normal	
StyleBox	read_only	

Signals

caret_changed() 🔗

Emitted when any caret changes position.

gutter_added() 🔗

Emitted when a gutter is added.

---

gutter_clicked(line: int, gutter: int)

Emitted when a gutter is clicked.

---

gutter_removed()

Emitted when a gutter is removed.

---

lines_edited_from(from_line: int, to_line: int)

Emitted immediately when the text changes.

When text is added `from_line` will be less than `to_line`. On a remove `to_line` will be less than `from_line`.

---

text_changed()

Emitted when the text changes.

---

text_set()

Emitted when clear is called or text is set.

Enumerations

enum **MenuItems**:

MenuItems **MENU_CUT** = `0`

Cuts (copies and clears) the selected text.

MenuItems **MENU_COPY** = `1`

Copies the selected text.

MenuItems **MENU_PASTE** = `2`

Pastes the clipboard text over the selected text (or at the cursor's position).

MenuItems **MENU_CLEAR** = `3`

Erases the whole **TextEdit** text.

MenuItems **MENU_SELECT_ALL** = `4`

Selects the whole **TextEdit** text.

MenuItems **MENU_UNDO** = `5`

Undoes the previous action.

MenuItems **MENU_REDO** = `6`

Redoes the previous action.

MenuItems **MENU_SUBMENU_TEXT_DIR** = `7`

ID of "Text Writing Direction" submenu.

MenuItems **MENU_DIR_INHERITED** = `8`

Sets text direction to inherited.

MenuItems **MENU_DIR_AUTO** = `9`

Sets text direction to automatic.

MenuItems **MENU_DIR_LTR** = `10`

Sets text direction to left-to-right.

MenuItems **MENU_DIR_RTL** = `11`

Sets text direction to right-to-left.

MenuItems **MENU_DISPLAY_UCC** = `12`

Toggles control character display.

MenuItems **MENU_SUBMENU_INSERT_UCC** = `13`

ID of "Insert Control Character" submenu.

MenuItems **MENU_INSERT_LRM** = `14`

Inserts left-to-right mark (LRM) character.

MenuItems **MENU_INSERT_RLM** = `15`

Inserts right-to-left mark (RLM) character.

MenuItems **MENU_INSERT_LRE** = `16`

Inserts start of left-to-right embedding (LRE) character.

MenuItems **MENU_INSERT_RLE** = `17`

Inserts start of right-to-left embedding (RLE) character.

MenuItems **MENU_INSERT_LRO** = `18`

Inserts start of left-to-right override (LRO) character.

MenuItems **MENU_INSERT_RLO** = `19`

Inserts start of right-to-left override (RLO) character.

MenuItems **MENU_INSERT_PDF** = `20`

Inserts pop direction formatting (PDF) character.

MenuItems **MENU_INSERT_ALM** = `21`

Inserts Arabic letter mark (ALM) character.

MenuItems **MENU_INSERT_LRI** = `22`

Inserts left-to-right isolate (LRI) character.

MenuItems **MENU_INSERT_RLI** = `23`

Inserts right-to-left isolate (RLI) character.

MenuItems **MENU_INSERT_FSI** = `24`

Inserts first strong isolate (FSI) character.

MenuItems **MENU_INSERT_PDI** = `25`

Inserts pop direction isolate (PDI) character.

MenuItems **MENU_INSERT_ZWJ** = `26`

Inserts zero width joiner (ZWJ) character.

MenuItems **MENU_INSERT_ZWNJ** = `27`

Inserts zero width non-joiner (ZWNJ) character.

MenuItems **MENU_INSERT_WJ** = `28`

Inserts word joiner (WJ) character.

MenuItems **MENU_INSERT_SHY** = `29`

Inserts soft hyphen (SHY) character.

MenuItems **MENU_MAX** = `30`

Represents the size of the MenuItems enum.

---

enum **EditAction**: 🔗

EditAction **ACTION_NONE** = `0`

No current action.

EditAction **ACTION_TYPING** = `1`

A typing action.

EditAction **ACTION_BACKSPACE** = `2`

A backwards delete action.

EditAction **ACTION_DELETE** = `3`

A forward delete action.

enum **SearchFlags**: 🔗

SearchFlags **SEARCH_MATCH_CASE** = `1`

Match case when searching.

SearchFlags **SEARCH_WHOLE_WORDS** = `2`

Match whole words when searching.

SearchFlags **SEARCH_BACKWARDS** = `4`

Search from end to beginning.

enum **CaretType**: 🔗

CaretType **CARET_TYPE_LINE** = `0`

Vertical line caret.

CaretType **CARET_TYPE_BLOCK** = `1`

Block caret.

---

enum **SelectionMode**: 🔗

SelectionMode **SELECTION_MODE_NONE** = `0`

Not selecting.

SelectionMode **SELECTION_MODE_SHIFT** = `1`

Select as if `shift` is pressed.

SelectionMode **SELECTION_MODE_POINTER** = `2`

Select single characters as if the user single clicked.

SelectionMode **SELECTION_MODE_WORD** = `3`

Select whole words as if the user double clicked.

SelectionMode **SELECTION_MODE_LINE** = `4`

Select whole lines as if the user triple clicked.

---

enum **LineWrappingMode**: 🔗

LineWrappingMode **LINE_WRAPPING_NONE** = `0`

Line wrapping is disabled.

LineWrappingMode **LINE_WRAPPING_BOUNDARY** = `1`

Line wrapping occurs at the control boundary, beyond what would normally be visible.

---

enum **GutterType**:

GutterType **GUTTER_TYPE_STRING** = `0`

When a gutter is set to string using set_gutter_type, it is used to contain text set via the set_line_gutter_text method.

GutterType **GUTTER_TYPE_ICON** = `1`

When a gutter is set to icon using set_gutter_type, it is used to contain an icon set via the set_line_gutter_icon method.

GutterType **GUTTER_TYPE_CUSTOM** = `2`

When a gutter is set to custom using set_gutter_type, it is used to contain custom visuals controlled by a callback method set via the set_gutter_custom_draw method.

Property Descriptions

AutowrapMode **autowrap_mode** = `3`

- void **set_autowrap_mode**(value: AutowrapMode)
- AutowrapMode **get_autowrap_mode**()

If wrap_mode is set to LINE_WRAPPING_BOUNDARY, sets text wrapping mode. To see how each mode behaves, see AutowrapMode.

---

bool **caret_blink** = `false`

- void **set_caret_blink_enabled**(value: bool)
- bool **is_caret_blink_enabled**()

If `true`, makes the caret blink.

---

float **caret_blink_interval** = `0.65` 🔗

- void **set_caret_blink_interval**(value: float)
- float **get_caret_blink_interval**()

The interval at which the caret blinks (in seconds).

---

bool **caret_draw_when_editable_disabled** = `false` 🔗

- void **set_draw_caret_when_editable_disabled**(value: bool)
- bool **is_drawing_caret_when_editable_disabled**()

If `true`, caret will be visible when editable is disabled.

---

bool **caret_mid_grapheme** = `false` 🔗

- void **set_caret_mid_grapheme_enabled**(value: bool)
- bool **is_caret_mid_grapheme_enabled**()

Allow moving caret, selecting and removing the individual composite character components.

Note: `Backspace` is always removing individual composite character components.

---

bool **caret_move_on_right_click** = `true` 🔗

- void **set_move_caret_on_right_click_enabled**(value: bool)
- bool **is_move_caret_on_right_click_enabled**()

If `true`, a right-click moves the caret at the mouse position before displaying the context menu.

If `false`, the context menu ignores mouse location.

---

bool **caret_multiple** = `true`

- void **set_multiple_carets_enabled**(value: bool)
- bool **is_multiple_carets_enabled**()

Sets if multiple carets are allowed.

CaretType **caret_type** = `0`

- void **set_caret_type**(value: CaretType)
- CaretType **get_caret_type**()

Set the type of caret to draw.

---

bool **context_menu_enabled** = `true`

- void **set_context_menu_enabled**(value: bool)
- bool **is_context_menu_enabled**()

If `true`, a right-click displays the context menu.

String **custom_word_separators** = `""`

- void **set_custom_word_separators**(value: String)
- String **get_custom_word_separators**()

The characters to consider as word delimiters if use_custom_word_separators is `true`. The characters should be defined without separation, for example `#_!`.

---

bool **deselect_on_focus_loss_enabled** = `true`

- void **set_deselect_on_focus_loss_enabled**(value: bool)
- bool **is_deselect_on_focus_loss_enabled**()

If `true`, the selected text will be deselected when focus is lost.

---

bool **drag_and_drop_selection_enabled** = `true`

- void **set_drag_and_drop_selection_enabled**(value: bool)
- bool **is_drag_and_drop_selection_enabled**()

If `true`, allow drag and drop of selected text. Text can still be dropped from other sources.

---

bool **draw_control_chars** = `false`

- void **set_draw_control_chars**(value: bool)
- bool **get_draw_control_chars**()

If `true`, control characters are displayed.

---

bool **draw_spaces** = `false`

- void **set_draw_spaces**(value: bool)
- bool **is_drawing_spaces**()

If `true`, the "space" character will have a visible representation.

---

bool **draw_tabs** = `false`

- void **set_draw_tabs**(value: bool)
- bool **is_drawing_tabs**()

If `true`, the "tab" character will have a visible representation.

---

bool **editable** = `true`

- void **set_editable**(value: bool)

- bool **is_editable**()

If `false`, existing text cannot be modified and new text cannot be added.

---

bool **highlight_all_occurrences** = `false`

- void **set_highlight_all_occurrences**(value: bool)
- bool **is_highlight_all_occurrences_enabled**()

If `true`, all occurrences of the selected text will be highlighted.

---

bool **highlight_current_line** = `false`

- void **set_highlight_current_line**(value: bool)
- bool **is_highlight_current_line_enabled**()

If `true`, the line containing the cursor is highlighted.

---

bool **indent_wrapped_lines** = `false`

- void **set_indent_wrapped_lines**(value: bool)
- bool **is_indent_wrapped_lines**()

If `true`, all wrapped lines are indented to the same amount as the unwrapped line.

String **language** = `""`

- void **set_language**(value: String)
- String **get_language**()

Language code used for line-breaking and text shaping algorithms, if left empty current locale is used instead.

---

bool **middle_mouse_paste_enabled** = `true` 🔗

- void **set_middle_mouse_paste_enabled**(value: bool)
- bool **is_middle_mouse_paste_enabled**()

If `false`, using middle mouse button to paste clipboard will be disabled.

Note: This method is only implemented on Linux.

---

bool **minimap_draw** = `false` 🔗

- void **set_draw_minimap**(value: bool)
- bool **is_drawing_minimap**()

If `true`, a minimap is shown, providing an outline of your source code. The minimap uses a fixed-width text size.

---

int **minimap_width** = `80` 🔗

- void **set_minimap_width**(value: int)
- int **get_minimap_width**()

The width, in pixels, of the minimap.

String **placeholder_text** = `""` 🔗

- void **set_placeholder**(value: String)
- String **get_placeholder**()

Text shown when the **TextEdit** is empty. It is **not** the **TextEdit**'s default value (see text).

---

bool **scroll_fit_content_height** = `false` 🔗

- void **set_fit_content_height_enabled**(value: bool)

- bool **is_fit_content_height_enabled**()

If `true`, **TextEdit** will disable vertical scroll and fit minimum height to the number of visible lines.

---

int **scroll_horizontal** = `0`

- void **set_h_scroll**(value: int)
- int **get_h_scroll**()

If there is a horizontal scrollbar, this determines the current horizontal scroll value in pixels.

---

bool **scroll_past_end_of_file** = `false`

- void **set_scroll_past_end_of_file_enabled**(value: bool)
- bool **is_scroll_past_end_of_file_enabled**()

Allow scrolling past the last line into "virtual" space.

---

bool **scroll_smooth** = `false`

- void **set_smooth_scroll_enabled**(value: bool)
- bool **is_smooth_scroll_enabled**()

Scroll smoothly over the text rather than jumping to the next location.

---

float **scroll_v_scroll_speed** = `80.0`

- void **set_v_scroll_speed**(value: float)
- float **get_v_scroll_speed**()

Sets the scroll speed with the minimap or when scroll_smooth is enabled.

---

float **scroll_vertical** = `0.0`

- void **set_v_scroll**(value: float)
- float **get_v_scroll**()

If there is a vertical scrollbar, this determines the current vertical scroll value in line numbers, starting at 0 for the top line.

bool **selecting_enabled** = `true`

- void **set_selecting_enabled**(value: bool)
- bool **is_selecting_enabled**()

If `true`, text can be selected.

If `false`, text can not be selected by the user or by the select or select_all methods.

bool **shortcut_keys_enabled** = `true`

- void **set_shortcut_keys_enabled**(value: bool)
- bool **is_shortcut_keys_enabled**()

If `true`, shortcut keys for context menu items are enabled, even if the context menu is disabled.

StructuredTextParser **structured_text_bidi_override** = `0`

- void **set_structured_text_bidi_override**(value: StructuredTextParser)
- StructuredTextParser **get_structured_text_bidi_override**()

Set BiDi algorithm override for the structured text.

Array **structured_text_bidi_override_options** = `[]`

- void **set_structured_text_bidi_override_options**(value: Array)
- Array **get_structured_text_bidi_override_options**()

Set additional options for BiDi override.

SyntaxHighlighter **syntax_highlighter** 🔗

- void **set_syntax_highlighter**(value: SyntaxHighlighter)
- SyntaxHighlighter **get_syntax_highlighter**()

Sets the SyntaxHighlighter to use.

String **text** = `""` 🔗

- void **set_text**(value: String)
- String **get_text**()

String value of the **TextEdit**.

TextDirection **text_direction** = `0` 🔗

- void **set_text_direction**(value: TextDirection)
- TextDirection **get_text_direction**()

Base text writing direction.

---

bool **use_custom_word_separators** = `false` 🔗

- void **set_use_custom_word_separators**(value: bool)
- bool **is_custom_word_separators_enabled**()

If `false`, using `Ctrl + Left` or `Ctrl + Right` (`Cmd + Left` or `Cmd + Right` on macOS) bindings will use the behavior of use_default_word_separators. If `true`, it will also stop the caret if a character within custom_word_separators is detected. Useful for subword moving. This behavior also will be applied to the behavior of text selection.

---

bool **use_default_word_separators** = `true` 🔗

- void **set_use_default_word_separators**(value: bool)
- bool **is_default_word_separators_enabled**()

If `false`, using `Ctrl + Left` or `Ctrl + Right` (`Cmd + Left` or `Cmd + Right` on macOS) bindings will stop moving caret only if a space or punctuation is detected. If `true`, it will also stop the caret if a character is part of `` !"#$%&'()*+,-./:;<=>?@[\]^`{|}~ ``, the Unicode General Punctuation table, or the Unicode CJK Punctuation table. Useful for subword moving. This behavior also will be applied to the behavior of text selection.

---

bool **virtual_keyboard_enabled** = `true` 🔗

- void **set_virtual_keyboard_enabled**(value: bool)
- bool **is_virtual_keyboard_enabled**()

If `true`, the native virtual keyboard is shown when focused on platforms that support it.

LineWrappingMode **wrap_mode** = `0` 🔗

- void **set_line_wrapping_mode**(value: LineWrappingMode)
- LineWrappingMode **get_line_wrapping_mode**()

Sets the line wrapping mode to use.

Method Descriptions

void **_backspace**(caret_index: int) virtual

Override this method to define what happens when the user presses the backspace key.

---

void **_copy**(caret_index: int) virtual

Override this method to define what happens when the user performs a copy operation.

---

void **_cut**(caret_index: int) virtual

Override this method to define what happens when the user performs a cut operation.

---

void **_handle_unicode_input**(unicode_char: int, caret_index: int) virtual

Override this method to define what happens when the user types in the provided key `unicode_char`.

---

void **_paste**(caret_index: int) virtual

Override this method to define what happens when the user performs a paste operation.

---

void **_paste_primary_clipboard**(caret_index: int) virtual

Override this method to define what happens when the user performs a paste operation with middle mouse button.

Note: This method is only implemented on Linux.

---

int **add_caret**(line: int, column: int) 🔗

Adds a new caret at the given location. Returns the index of the new caret, or `-1` if the location is invalid.

---

void **add_caret_at_carets**(below: bool) 🔗

Adds an additional caret above or below every caret. If `below` is `true` the new caret will be added below and above otherwise.

---

void **add_gutter**(at: int = -1) 🔗

Register a new gutter to this **TextEdit**. Use `at` to have a specific gutter order. A value of `-1` appends the gutter to the right.

---

void **add_selection_for_next_occurrence**() 🔗

Adds a selection and a caret for the next occurrence of the current selection. If there is no active selection, selects word under caret.

---

void **adjust_carets_after_edit**(caret: int, from_line: int, from_col: int, to_line: int, to_col: int) 🔗

Deprecated: No longer necessary since methods now adjust carets themselves.

This method does nothing.

---

void **adjust_viewport_to_caret**(caret_index: int = 0) 🔗

Adjust the viewport so the caret is visible.

---

void **apply_ime**()

Applies text from the Input Method Editor [https://en.wikipedia.org/wiki/Input_method] (IME) to each caret and closes the IME if it is open.

---

void **backspace**(caret_index: int = -1)

Called when the user presses the backspace key. Can be overridden with _backspace.

---

void **begin_complex_operation**()

Starts a multipart edit. All edits will be treated as one action until end_complex_operation is called.

---

void **begin_multicaret_edit**()

Starts an edit for multiple carets. The edit must be ended with end_multicaret_edit. Multicaret edits can be used to edit text at multiple carets and delay merging the carets until the end, so the caret indexes aren't affected immediately. begin_multicaret_edit and end_multicaret_edit can be nested, and the merge will happen at the last end_multicaret_edit.

Example usage:

```
begin_complex_operation()
begin_multicaret_edit()
for i in range(get_caret_count()):
    if multicaret_edit_ignore_caret(i):
        continue
    # Logic here.
end_multicaret_edit()
end_complex_operation()
```

---

void **cancel_ime**()

Closes the Input Method Editor [https://en.wikipedia.org/wiki/Input_method] (IME) if it is open. Any text in the IME will be lost.

void **center_viewport_to_caret**(caret_index: int = 0)

Centers the viewport on the line the editing caret is at. This also resets the scroll_horizontal value to `0`.

void **clear**()

Performs a full reset of **TextEdit**, including undo history.

void **clear_undo_history**()

Clears the undo history.

void **collapse_carets**(from_line: int, from_column: int, to_line: int, to_column: int, inclusive: bool = false)

Collapse all carets in the given range to the `from_line` and `from_column` position.

`inclusive` applies to both ends.

If is_in_mulitcaret_edit is `true`, carets that are collapsed will be `true` for multicaret_edit_ignore_caret.

merge_overlapping_carets will be called if any carets were collapsed.

void **copy**(caret_index: int = -1)

Copies the current text selection. Can be overridden with _copy.

---

void **cut**(caret_index: int = -1)

Cut's the current selection. Can be overridden with _cut.

---

void **delete_selection**(caret_index: int = -1)

Deletes the selected text.

---

void **deselect**(caret_index: int = -1)

Deselects the current selection.

---

void **end_action**()

Marks the end of steps in the current action started with start_action.

---

void **end_complex_operation**()

Ends a multipart edit, started with begin_complex_operation. If called outside a complex operation, the current operation is pushed onto the undo/redo stack.

---

void **end_multicaret_edit**()

Ends an edit for multiple carets, that was started with begin_multicaret_edit. If this was the last end_multicaret_edit and merge_overlapping_carets was called, carets will be merged.

---

int **get_caret_column**(caret_index: int = 0) const

Returns the column the editing caret is at.

---

int **get_caret_count**() const

Returns the number of carets in this **TextEdit**.

Vector2 **get_caret_draw_pos**(caret_index: int = 0) const

Returns the caret pixel draw position.

PackedInt32Array **get_caret_index_edit_order**()

Deprecated: Carets no longer need to be edited in any specific order. If the carets need to be sorted, use get_sorted_carets instead.

Returns a list of caret indexes in their edit order, this done from bottom to top. Edit order refers to the way actions such as insert_text_at_caret are applied.

---

int **get_caret_line**(caret_index: int = 0) const

Returns the line the editing caret is on.

---

int **get_caret_wrap_index**(caret_index: int = 0) const

Returns the wrap index the editing caret is on.

---

int **get_first_non_whitespace_column**(line: int) const

Returns the first column containing a non-whitespace character.

---

int **get_first_visible_line**() const

Returns the first visible line.

int **get_gutter_count**() const

Returns the number of gutters registered.

String **get_gutter_name**(gutter: int) const

Returns the name of the gutter at the given index.

GutterType **get_gutter_type**(gutter: int) const

Returns the type of the gutter at the given index. Gutters can contain icons, text, or custom visuals. See GutterType for options.

int **get_gutter_width**(gutter: int) const

Returns the width of the gutter at the given index.

HScrollBar **get_h_scroll_bar**() const

Returns the HScrollBar used by **TextEdit**.

int **get_indent_level**(line: int) const

Returns the number of spaces and `tab * tab_size` before the first char.

int **get_last_full_visible_line**() const

Returns the last visible line. Use get_last_full_visible_line_wrap_index for the wrap index.

int **get_last_full_visible_line_wrap_index**() const 🔗

Returns the last visible wrap index of the last visible line.

---

int **get_last_unhidden_line**() const 🔗

Returns the last unhidden line in the entire **TextEdit**.

String **get_line**(line: int) const 🔗

Returns the text of a specific line.

Color **get_line_background_color**(line: int) const 🔗

Returns the current background color of the line. `Color(0, 0, 0, 0)` is returned if no color is set.

Vector2i **get_line_column_at_pos**(position: Vector2i, allow_out_of_bounds: bool = true) const 🔗

Returns the line and column at the given position. In the returned vector, `x` is the column, `y` is the line. If `allow_out_of_bounds` is `false` and the position is not over the text, both vector values will be set to `-1`.

---

int **get_line_count**() const 🔗

Returns the number of lines in the text.

Texture2D **get_line_gutter_icon**(line: int, gutter: int) const 🔗

Returns the icon currently in `gutter` at `line`. This only works when the gutter type is GUTTER_TYPE_ICON (see set_gutter_type).

Color **get_line_gutter_item_color**(line: int, gutter: int) const

Returns the color currently in `gutter` at `line`.

Variant **get_line_gutter_metadata**(line: int, gutter: int) const

Returns the metadata currently in `gutter` at `line`.

String **get_line_gutter_text**(line: int, gutter: int) const

Returns the text currently in `gutter` at `line`. This only works when the gutter type is GUTTER_TYPE_STRING (see set_gutter_type).

int **get_line_height**() const

Returns the maximum value of the line height among all lines.

Note: The return value is influenced by line_spacing and font_size. And it will not be less than `1`.

Array[Vector2i] **get_line_ranges_from_carets**(only_selections: bool = false, merge_adjacent: bool = true) const

Returns an Array of line ranges where `x` is the first line and `y` is the last line. All lines within these ranges will have a caret on them or be part of a selection. Each line will only be part of one line range, even if it has multiple carets on it.

If a selection's end column (get_selection_to_column) is at column `0`, that line will not be included. If a selection begins on the line after another selection ends and `merge_adjacent` is `true`, or they begin and end on the same line, one line range will include both selections.

int **get_line_width**(line: int, wrap_index: int = -1) const

Returns the width in pixels of the `wrap_index` on `line`.

int **get_line_wrap_count**(line: int) const

Returns the number of times the given line is wrapped.

int **get_line_wrap_index_at_column**(line: int, column: int) const

Returns the wrap index of the given line column.

PackedStringArray **get_line_wrapped_text**(line: int) const

Returns an array of Strings representing each wrapped index.

Vector2 **get_local_mouse_pos**() const

Returns the local mouse position adjusted for the text direction.

PopupMenu **get_menu**() const

Returns the PopupMenu of this **TextEdit**. By default, this menu is displayed when right-clicking on the **TextEdit**.

You can add custom menu items or remove standard ones. Make sure your IDs don't conflict with the standard ones (see MenuItems). For example:

GDScript

```gdscript
func _ready():
    var menu = get_menu()
    # Remove all items after "Redo".
    menu.item_count = menu.get_item_index(MENU_REDO) + 1
    # Add custom items.
    menu.add_separator()
    menu.add_item("Insert Date", MENU_MAX + 1)
    # Connect callback.
    menu.id_pressed.connect(_on_item_pressed)

func _on_item_pressed(id):
    if id == MENU_MAX + 1:
        insert_text_at_caret(Time.get_date_string_from_system())
```

C#

```csharp
public override void _Ready()
{
    var menu = GetMenu();
    // Remove all items after "Redo".
    menu.ItemCount = menu.GetItemIndex(TextEdit.MenuItems.Redo) +
1;
    // Add custom items.
    menu.AddSeparator();
    menu.AddItem("Insert Date", TextEdit.MenuItems.Max + 1);
    // Add event handler.
    menu.IdPressed += OnItemPressed;
}

public void OnItemPressed(int id)
{
    if (id == TextEdit.MenuItems.Max + 1)
    {
        InsertTextAtCaret(Time.GetDateStringFromSystem());
    }
}
```

Warning: This is a required internal node, removing and freeing it may cause a crash. If you wish to hide it or any of its children, use their Window.visible property.

int **get_minimap_line_at_pos**(position: Vector2i) const 🔗

Returns the equivalent minimap line at `position`.

---

int **get_minimap_visible_lines**() const 🔗

Returns the number of lines that may be drawn on the minimap.

Vector2i **get_next_visible_line_index_offset_from**(line: int, wrap_index: int, visible_amount: int) const 🔗

Similar to get_next_visible_line_offset_from, but takes into account the line wrap indexes. In the returned vector, `x` is the line, `y` is the wrap index.

---

int **get_next_visible_line_offset_from**(line: int, visible_amount: int) const 🔗

Returns the count to the next visible line from `line` to `line + visible_amount`. Can also count backwards. For example if a **TextEdit** has 5 lines with lines 2 and 3 hidden, calling this with `line = 1, visible_amount = 1` would return 3.

Vector2i **get_pos_at_line_column**(line: int, column: int) const 🔗

Returns the local position for the given `line` and `column`. If `x` or `y` of the returned vector equal `-1`, the position is outside of the viewable area of the control.

Note: The Y position corresponds to the bottom side of the line. Use get_rect_at_line_column to get the top side position.

Rect2i **get_rect_at_line_column**(line: int, column: int) const 🔗

Returns the local position and size for the grapheme at the given `line` and `column`. If `x` or `y` position of the returned rect equal `-1`, the position is outside of the viewable area of the control.

Note: The Y position of the returned rect corresponds to the top side of the line, unlike get_pos_at_line_column which returns the bottom side.

---

int **get_saved_version**() const

Returns the last tagged saved version from tag_saved_version.

---

float **get_scroll_pos_for_line**(line: int, wrap_index: int = 0) const

Returns the scroll position for `wrap_index` of `line`.

String **get_selected_text**(caret_index: int = -1)

Returns the text inside the selection of a caret, or all the carets if `caret_index` is its default value `-1`.

---

int **get_selection_at_line_column**(line: int, column: int, include_edges: bool = true, only_selections: bool = true) const

Returns the caret index of the selection at the given `line` and `column`, or `-1` if there is none.

If `include_edges` is `false`, the position must be inside the selection and not at either end. If `only_selections` is `false`, carets without a selection will also be considered.

---

int **get_selection_column**(caret_index: int = 0) const

Deprecated: Use get_selection_origin_column instead.

Returns the original start column of the selection.

int **get_selection_from_column**(caret_index: int = 0) const

Returns the selection begin column. Returns the caret column if there is no selection.

int **get_selection_from_line**(caret_index: int = 0) const

Returns the selection begin line. Returns the caret line if there is no selection.

int **get_selection_line**(caret_index: int = 0) const

Deprecated: Use get_selection_origin_line instead.

Returns the original start line of the selection.

SelectionMode **get_selection_mode**() const

Returns the current selection mode.

int **get_selection_origin_column**(caret_index: int = 0) const

Returns the origin column of the selection. This is the opposite end from the caret.

int **get_selection_origin_line**(caret_index: int = 0) const

Returns the origin line of the selection. This is the opposite end from the caret.

---

int **get_selection_to_column**(caret_index: int = 0) const 🔗

Returns the selection end column. Returns the caret column if there is no selection.

---

int **get_selection_to_line**(caret_index: int = 0) const 🔗

Returns the selection end line. Returns the caret line if there is no selection.

PackedInt32Array **get_sorted_carets**(include_ignored_carets: bool = false) const 🔗

Returns the carets sorted by selection beginning from lowest line and column to highest (from top to bottom of text).

If `include_ignored_carets` is `false`, carets from multicaret_edit_ignore_caret will be ignored.

---

int **get_tab_size**() const 🔗

Returns the **TextEdit**'s' tab size.

---

int **get_total_gutter_width**() const 🔗

Returns the total width of all gutters and internal padding.

---

int **get_total_visible_line_count**() const 🔗

Returns the number of lines that may be drawn.

VScrollBar **get_v_scroll_bar**() const

Returns the VScrollBar of the **TextEdit**.

---

int **get_version**() const

Returns the current version of the **TextEdit**. The version is a count of recorded operations by the undo/redo history.

---

int **get_visible_line_count**() const

Returns the number of visible lines, including wrapped text.

---

int **get_visible_line_count_in_range**(from_line: int, to_line: int) const

Returns the total number of visible + wrapped lines between the two lines.

String **get_word_at_pos**(position: Vector2) const

Returns the word at `position`.

String **get_word_under_caret**(caret_index: int = -1) const

Returns a String text with the word under the caret's location.

---

bool **has_ime_text**() const

Returns `true` if the user has text in the Input Method Editor [https://en.wikipedia.org/wiki/Input_method] (IME).

---

bool **has_redo**() const

Returns `true` if a "redo" action is available.

---

bool **has_selection**(caret_index: int = -1) const

Returns `true` if the user has selected text.

---

bool **has_undo**() const

Returns `true` if an "undo" action is available.

---

void **insert_line_at**(line: int, text: String)

Inserts a new line with `text` at `line`.

---

void **insert_text**(text: String, line: int, column: int, before_selection_begin: bool = true, before_selection_end: bool = false)

Inserts the `text` at `line` and `column`.

If `before_selection_begin` is `true`, carets and selections that begin at `line` and `column` will moved to the end of the inserted text, along with all carets after it.

If `before_selection_end` is `true`, selections that end at `line` and `column` will be extended to the end of the inserted text. These parameters can be used to insert text inside of or outside of selections.

---

void **insert_text_at_caret**(text: String, caret_index: int = -1)

Insert the specified text at the caret position.

bool **is_caret_after_selection_origin**(caret_index: int = 0) const 🔗

Returns `true` if the caret of the selection is after the selection origin. This can be used to determine the direction of the selection.

bool **is_caret_visible**(caret_index: int = 0) const 🔗

Returns `true` if the caret is visible on the screen.

bool **is_dragging_cursor**() const 🔗

Returns `true` if the user is dragging their mouse for scrolling, selecting, or text dragging.

bool **is_gutter_clickable**(gutter: int) const 🔗

Returns whether the gutter is clickable.

bool **is_gutter_drawn**(gutter: int) const 🔗

Returns whether the gutter is currently drawn.

bool **is_gutter_overwritable**(gutter: int) const 🔗

Returns whether the gutter is overwritable.

bool **is_in_mulitcaret_edit**() const 🔗

Returns `true` if a begin_multicaret_edit has been called and end_multicaret_edit has not yet been called.

bool **is_line_gutter_clickable**(line: int, gutter: int) const 🔗

Returns whether the gutter on the given line is clickable.

---

bool **is_line_wrapped**(line: int) const 🔗

Returns if the given line is wrapped.

---

bool **is_menu_visible**() const 🔗

Returns whether the menu is visible. Use this instead of `get_menu().visible` to improve performance (so the creation of the menu is avoided).

---

bool **is_mouse_over_selection**(edges: bool, caret_index: int = -1) const 🔗

Returns whether the mouse is over selection. If `edges` is `true`, the edges are considered part of the selection.

---

bool **is_overtype_mode_enabled**() const 🔗

Returns whether the user is in overtype mode.

---

void **menu_option**(option: int) 🔗

Executes a given action as defined in the MenuItems enum.

---

void **merge_gutters**(from_line: int, to_line: int) 🔗

Merge the gutters from `from_line` into `to_line`. Only overwritable gutters will be copied.

---

void **merge_overlapping_carets**() 🔗

Merges any overlapping carets. Will favor the newest caret, or the caret with a selection.

If is_in_mulitcaret_edit is `true`, the merge will be queued to happen at the end of the multicaret edit. See begin_multicaret_edit and end_multicaret_edit.

Note: This is not called when a caret changes position but after certain actions, so it is possible to get into a state where carets overlap.

---

bool **multicaret_edit_ignore_caret**(caret_index: int) const 🔗

Returns `true` if the given `caret_index` should be ignored as part of a multicaret edit. See begin_multicaret_edit and end_multicaret_edit. Carets that should be ignored are ones that were part of removed text and will likely be merged at the end of the edit, or carets that were added during the edit.

It is recommended to `continue` within a loop iterating on multiple carets if a caret should be ignored.

---

void **paste**(caret_index: int = -1) 🔗

Paste at the current location. Can be overridden with _paste.

---

void **paste_primary_clipboard**(caret_index: int = -1) 🔗

Pastes the primary clipboard.

---

void **redo**() 🔗

Perform redo operation.

---

void **remove_caret**(caret: int)

Removes the given caret index.

Note: This can result in adjustment of all other caret indices.

---

void **remove_gutter**(gutter: int)

Removes the gutter from this **TextEdit**.

---

void **remove_line_at**(line: int, move_carets_down: bool = true)

Removes the line of text at `line`. Carets on this line will attempt to match their previous visual x position.

If `move_carets_down` is `true` carets will move to the next line down, otherwise carets will move up.

---

void **remove_secondary_carets**()

Removes all additional carets.

---

void **remove_text**(from_line: int, from_column: int, to_line: int, to_column: int)

Removes text between the given positions.

Vector2i **search**(text: String, flags: int, from_line: int, from_column: int) const

Perform a search inside the text. Search flags can be specified in the SearchFlags enum.

In the returned vector, `x` is the column, `y` is the line. If no results are found, both are equal to `-1`.

GDScript

```
var result = search("print", SEARCH_WHOLE_WORDS, 0, 0)
if result.x != -1:
    # Result found.
    var line_number = result.y
    var column_number = result.x
```

C#

```
Vector2I result = Search("print",
(uint)TextEdit.SearchFlags.WholeWords, 0, 0);
if (result.X != -1)
{
    // Result found.
    int lineNumber = result.Y;
    int columnNumber = result.X;
}
```

---

void **select**(origin_line: int, origin_column: int, caret_line: int, caret_column: int, caret_index: int = 0)

Selects text from `origin_line` and `origin_column` to `caret_line` and `caret_column` for the given `caret_index`. This moves the selection origin and the caret. If the positions are the same, the selection will be deselected.

If selecting_enabled is `false`, no selection will occur.

Note: If supporting multiple carets this will not check for any overlap. See merge_overlapping_carets.

---

void **select_all**()

Select all the text.

If selecting_enabled is `false`, no selection will occur.

---

void **select_word_under_caret**(caret_index: int = -1)

Selects the word under the caret.

---

void **set_caret_column**(column: int, adjust_viewport: bool = true, caret_index: int = 0)

Moves the caret to the specified `column` index.

If `adjust_viewport` is `true`, the viewport will center at the caret position after the move occurs.

Note: If supporting multiple carets this will not check for any overlap. See merge_overlapping_carets.

---

void **set_caret_line**(line: int, adjust_viewport: bool = true, can_be_hidden: bool = true, wrap_index: int = 0, caret_index: int = 0)

Moves the caret to the specified `line` index. The caret column will be moved to the same visual position it was at the last time set_caret_column was called, or clamped to the end of the line.

If `adjust_viewport` is `true`, the viewport will center at the caret position after the move occurs.

If `can_be_hidden` is `true`, the specified `line` can be hidden.

If `wrap_index` is `-1`, the caret column will be clamped to the `line`'s length. If `wrap_index` is greater than `-1`, the column will be moved to attempt to match the visual x position on the line's `wrap_index` to the position from the last time set_caret_column was called.

Note: If supporting multiple carets this will not check for any overlap. See merge_overlapping_carets.

---

void **set_gutter_clickable**(gutter: int, clickable: bool)

Sets the gutter as clickable. This will change the mouse cursor to a pointing hand when hovering over the gutter.

---

void **set_gutter_custom_draw**(column: int, draw_callback: Callable)

Set a custom draw method for the gutter. The callback method must take the following args: `line: int, gutter: int, Area: Rect2`. This only works when the gutter type is GUTTER_TYPE_CUSTOM (see set_gutter_type).

---

void **set_gutter_draw**(gutter: int, draw: bool)

Sets whether the gutter should be drawn.

---

void **set_gutter_name**(gutter: int, name: String)

Sets the name of the gutter.

---

void **set_gutter_overwritable**(gutter: int, overwritable: bool)

Sets the gutter to overwritable. See merge_gutters.

---

void **set_gutter_type**(gutter: int, type: GutterType)

Sets the type of gutter. Gutters can contain icons, text, or custom visuals. See GutterType for options.

---

void **set_gutter_width**(gutter: int, width: int)

Set the width of the gutter.

---

void **set_line**(line: int, new_text: String)

Sets the text for a specific `line`.

Carets on the line will attempt to keep their visual x position.

---

void **set_line_as_center_visible**(line: int, wrap_index: int = 0)

Positions the `wrap_index` of `line` at the center of the viewport.

---

void **set_line_as_first_visible**(line: int, wrap_index: int = 0)

Positions the `wrap_index` of `line` at the top of the viewport.

---

void **set_line_as_last_visible**(line: int, wrap_index: int = 0)

Positions the `wrap_index` of `line` at the bottom of the viewport.

---

void **set_line_background_color**(line: int, color: Color)

Sets the current background color of the line. Set to `Color(0, 0, 0, 0)` for no color.

---

void **set_line_gutter_clickable**(line: int, gutter: int, clickable: bool) 🔗

If `clickable` is `true`, makes the `gutter` on `line` clickable. See gutter_clicked.

---

void **set_line_gutter_icon**(line: int, gutter: int, icon: Texture2D) 🔗

Sets the icon for `gutter` on `line` to `icon`. This only works when the gutter type is GUTTER_TYPE_ICON (see set_gutter_type).

---

void **set_line_gutter_item_color**(line: int, gutter: int, color: Color) 🔗

Sets the color for `gutter` on `line` to `color`.

---

void **set_line_gutter_metadata**(line: int, gutter: int, metadata: Variant) 🔗

Sets the metadata for `gutter` on `line` to `metadata`.

---

void **set_line_gutter_text**(line: int, gutter: int, text: String) 🔗

Sets the text for `gutter` on `line` to `text`. This only works when the gutter type is GUTTER_TYPE_STRING (see set_gutter_type).

---

void **set_overtype_mode_enabled**(enabled: bool) 🔗

If `true`, sets the user into overtype mode. When the user types in this mode, it will override existing text.

---

void **set_search_flags**(flags: int) 🔗

Sets the search `flags`. This is used with set_search_text to highlight occurrences of the searched text. Search flags can be specified from the SearchFlags enum.

---

void **set_search_text**(search_text: String)

Sets the search text. See set_search_flags.

---

void **set_selection_mode**(mode: SelectionMode)

Sets the current selection mode.

---

void **set_selection_origin_column**(column: int, caret_index: int = 0)

Sets the selection origin column to the `column` for the given `caret_index`. If the selection origin is moved to the caret position, the selection will deselect.

---

void **set_selection_origin_line**(line: int, can_be_hidden: bool = true, wrap_index: int = -1, caret_index: int = 0)

Sets the selection origin line to the `line` for the given `caret_index`. If the selection origin is moved to the caret position, the selection will deselect.

If `can_be_hidden` is `false`, The line will be set to the nearest unhidden line below or above.

If `wrap_index` is `-1`, the selection origin column will be clamped to the `line`'s length. If `wrap_index` is greater than `-1`, the column will be moved to attempt to match the visual x position on the line's `wrap_index` to the position from the last time set_selection_origin_column or select was called.

---

void **set_tab_size**(size: int) 🔗

Sets the tab size for the **TextEdit** to use.

---

void **set_tooltip_request_func**(callback: Callable) 🔗

Provide custom tooltip text. The callback method must take the following args: `hovered_word: String`.

---

void **skip_selection_for_next_occurrence**() 🔗

Moves a selection and a caret for the next occurrence of the current selection. If there is no active selection, moves to the next occurrence of the word under caret.

---

void **start_action**(action: EditAction) 🔗

Starts an action, will end the current action if `action` is different.

An action will also end after a call to end_action, after ProjectSettings.gui/timers/text_edit_idle_detect_sec is triggered or a new undoable step outside the start_action and end_action calls.

---

void **swap_lines**(from_line: int, to_line: int) 🔗

Swaps the two lines. Carets will be swapped with the lines.

---

void **tag_saved_version**() 🔗

Tag the current version as saved.

---

void **undo**()

Perform undo operation.

Theme Property Descriptions

Color **background_color** = `Color(0, 0, 0, 0)`

Sets the background Color of this **TextEdit**.

Color **caret_background_color** = `Color(0, 0, 0, 1)`

Color of the text behind the caret when using a block caret.

Color **caret_color** = `Color(0.875, 0.875, 0.875, 1)`

Color of the caret. This can be set to a fully transparent color to hide the caret entirely.

Color **current_line_color** = `Color(0.25, 0.25, 0.26, 0.8)`

Background Color of the line containing the caret.

Color **font_color** = `Color(0.875, 0.875, 0.875, 1)`

Sets the font Color.

Color **font_outline_color** = `Color(0, 0, 0, 1)`

The tint of text outline of the **TextEdit**.

Color **font_placeholder_color** = `Color(0.875, 0.875, 0.875, 0.6)`

Font color for placeholder_text.

Color **font_readonly_color** = `Color(0.875, 0.875, 0.875, 0.5)`

Sets the font Color when editable is disabled.

Color **font_selected_color** = `Color(0, 0, 0, 0)`

Sets the Color of the selected text. If equal to `Color(0, 0, 0, 0)`, it will be ignored.

Color **search_result_border_color** = `Color(0.3, 0.3, 0.3, 0.4)`

Color of the border around text that matches the search query.

Color **search_result_color** = `Color(0.3, 0.3, 0.3, 1)`

Color behind the text that matches the search query.

Color **selection_color** = `Color(0.5, 0.5, 0.5, 1)`

Sets the highlight Color of text selections.

Color **word_highlighted_color** = `Color(0.5, 0.5, 0.5, 0.25)`

Sets the highlight Color of multiple occurrences. highlight_all_occurrences has to be enabled.

int **caret_width** = `1` 🔗

The caret's width in pixels. Greater values can be used to improve accessibility by ensuring the caret is easily visible, or to ensure consistency with a large font size. If set to `0` or lower, the caret width is automatically set to 1 pixel and multiplied by the display scaling factor.

int **line_spacing** = `4` 🔗

Sets the spacing between the lines.

int **outline_size** = `0` 🔗

The size of the text outline.

Note: If using a font with FontFile.multichannel_signed_distance_field enabled, its FontFile.msdf_pixel_range must be set to at least *twice* the value of outline_size for outline rendering to look correct. Otherwise, the outline may appear to be cut off earlier than intended.

Font **font** 🔗

Sets the default Font.

int **font_size** 🔗

Sets default font size.

Texture2D **space** 🔗

Sets a custom Texture2D for space text characters.

Texture2D **tab**

Sets a custom Texture2D for tab text characters.

StyleBox **focus**

Sets the StyleBox when in focus. The focus StyleBox is displayed *over* the base StyleBox, so a partially transparent StyleBox should be used to ensure the base StyleBox remains visible. A StyleBox that represents an outline or an underline works well for this purpose. To disable the focus visual effect, assign a StyleBoxEmpty resource. Note that disabling the focus visual effect will harm keyboard/controller navigation usability, so this is not recommended for accessibility reasons.

StyleBox **normal**

Sets the StyleBox of this **TextEdit**.

StyleBox **read_only**

Sets the StyleBox of this **TextEdit** when editable is disabled.

TextureButton

Inherits: BaseButton **<** Control **<** CanvasItem **<** Node **<** Object

Texture-based button. Supports Pressed, Hover, Disabled and Focused states.

Description

TextureButton has the same functionality as Button, except it uses sprites instead of Godot's Theme resource. It is faster to create, but it doesn't support localization like more complex Controls.

The "normal" state must contain a texture (texture_normal); other textures are optional.

See also BaseButton which contains common properties and methods associated with this node.

Tutorials

- 3D Voxel Demo [https://godotengine.org/asset-library/asset/2755]

Properties

bool	flip_h	`false`
bool	flip_v	`false`

bool	ignore_texture_size	`false`
StretchMode	stretch_mode	`2`
BitMap	texture_click_mask	
Texture2D	texture_disabled	
Texture2D	texture_focused	
Texture2D	texture_hover	
Texture2D	texture_normal	
Texture2D	texture_pressed	

Enumerations

enum **StretchMode**: 🔗

StretchMode **STRETCH_SCALE** = `0`

Scale to fit the node's bounding rectangle.

StretchMode **STRETCH_TILE** = `1`

Tile inside the node's bounding rectangle.

StretchMode **STRETCH_KEEP** = `2`

The texture keeps its original size and stays in the bounding rectangle's top-left corner.

StretchMode **STRETCH_KEEP_CENTERED** = `3`

The texture keeps its original size and stays centered in the node's bounding rectangle.

StretchMode **STRETCH_KEEP_ASPECT** = `4`

Scale the texture to fit the node's bounding rectangle, but maintain the texture's aspect ratio.

StretchMode **STRETCH_KEEP_ASPECT_CENTERED** = `5`

Scale the texture to fit the node's bounding rectangle, center it, and maintain its aspect ratio.

StretchMode **STRETCH_KEEP_ASPECT_COVERED** = `6`

Scale the texture so that the shorter side fits the bounding rectangle. The other side clips to the node's limits.

Property Descriptions

bool **flip_h** = `false` 🔗

- void **set_flip_h**(value: bool)
- bool **is_flipped_h**()

If `true`, texture is flipped horizontally.

bool **flip_v** = `false`

- void **set_flip_v**(value: bool)
- bool **is_flipped_v**()

If `true`, texture is flipped vertically.

bool **ignore_texture_size** = `false`

- void **set_ignore_texture_size**(value: bool)
- bool **get_ignore_texture_size**()

If `true`, the size of the texture won't be considered for minimum size calculation, so the **TextureButton** can be shrunk down past the texture size.

StretchMode **stretch_mode** = `2`

- void **set_stretch_mode**(value: StretchMode)
- StretchMode **get_stretch_mode**()

Controls the texture's behavior when you resize the node's bounding rectangle. See the StretchMode constants for available options.

BitMap **texture_click_mask**

- void **set_click_mask**(value: BitMap)
- BitMap **get_click_mask**()

Pure black and white BitMap image to use for click detection. On the mask, white pixels represent the button's clickable area. Use it to create buttons with curved shapes.

Texture2D **texture_disabled**

- void **set_texture_disabled**(value: Texture2D)
- Texture2D **get_texture_disabled**()

Texture to display when the node is disabled. See BaseButton.disabled.

Texture2D **texture_focused**

- void **set_texture_focused**(value: Texture2D)
- Texture2D **get_texture_focused**()

Texture to display when the node has mouse or keyboard focus. texture_focused is displayed *over* the base texture, so a partially transparent texture should be used to ensure the base texture remains visible. A texture that represents an outline or an underline works well for this purpose. To disable the focus visual effect, assign a fully transparent texture of any size. Note that disabling the focus visual effect will harm keyboard/controller navigation usability, so this is not recommended for accessibility reasons.

Texture2D **texture_hover**

- void **set_texture_hover**(value: Texture2D)
- Texture2D **get_texture_hover**()

Texture to display when the mouse hovers the node.

Texture2D **texture_normal**

- void **set_texture_normal**(value: Texture2D)
- Texture2D **get_texture_normal**()

Texture to display by default, when the node is **not** in the disabled, hover or pressed state. This texture is still displayed in the focused state, with texture_focused drawn on top.

Texture2D **texture_pressed** 🔗

- void **set_texture_pressed**(value: Texture2D)
- Texture2D **get_texture_pressed**()

Texture to display on mouse down over the node, if the node has keyboard focus and the player presses the Enter key or if the player presses the BaseButton.shortcut key.

TextureProgressBar

Inherits: Range **<** Control **<** CanvasItem **<** Node **<** Object

Texture-based progress bar. Useful for loading screens and life or stamina bars.

Description

TextureProgressBar works like ProgressBar, but uses up to 3 textures instead of Godot's Theme resource. It can be used to create horizontal, vertical and radial progress bars.

Properties

int	fill_mode	`0`
MouseFilter	mouse_filter	`1` (overrides Control)
bool	nine_patch_stretch	`false`
Vector2	radial_center_offset	`Vector2(0, 0)`
float	radial_fill_degrees	`360.0`
float	radial_initial_angle	`0.0`

BitField[SizeFlags]	size_flags_vertical	`1` (overrides Control)
float	step	`1.0` (overrides Range)
int	stretch_margin_bottom	`0`
int	stretch_margin_left	`0`
int	stretch_margin_right	`0`
int	stretch_margin_top	`0`
Texture2D	texture_over	
Texture2D	texture_progress	
Vector2	texture_progress_offset	`Vector2(0, 0)`
Texture2D	texture_under	
Color	tint_over	`Color(1, 1, 1, 1)`
Color	tint_progress	`Color(1, 1, 1, 1)`
Color	tint_under	`Color(1, 1, 1, 1)`

Methods

int	get_stretch_margin(margin: Side) const
void	set_stretch_margin(margin: Side, value: int)

Enumerations

enum **FillMode**: 🔗

FillMode **FILL_LEFT_TO_RIGHT** = `0`

The texture_progress fills from left to right.

FillMode **FILL_RIGHT_TO_LEFT** = `1`

The texture_progress fills from right to left.

FillMode **FILL_TOP_TO_BOTTOM** = `2`

The texture_progress fills from top to bottom.

FillMode **FILL_BOTTOM_TO_TOP** = `3`

The texture_progress fills from bottom to top.

FillMode **FILL_CLOCKWISE** = `4`

Turns the node into a radial bar. The texture_progress fills clockwise. See radial_center_offset, radial_initial_angle and radial_fill_degrees to control the way the bar fills up.

FillMode **FILL_COUNTER_CLOCKWISE** = 5

Turns the node into a radial bar. The texture_progress fills counterclockwise. See radial_center_offset, radial_initial_angle and radial_fill_degrees to control the way the bar fills up.

FillMode **FILL_BILINEAR_LEFT_AND_RIGHT** = 6

The texture_progress fills from the center, expanding both towards the left and the right.

FillMode **FILL_BILINEAR_TOP_AND_BOTTOM** = 7

The texture_progress fills from the center, expanding both towards the top and the bottom.

FillMode **FILL_CLOCKWISE_AND_COUNTER_CLOCKWISE** = 8

Turns the node into a radial bar. The texture_progress fills radially from the center, expanding both clockwise and counterclockwise. See radial_center_offset, radial_initial_angle and radial_fill_degrees to control the way the bar fills up.

Property Descriptions

int **fill_mode** = 0 🔗

- void **set_fill_mode**(value: int)
- int **get_fill_mode**()

The fill direction. See FillMode for possible values.

bool **nine_patch_stretch** = `false` 🔗

- void **set_nine_patch_stretch**(value: bool)
- bool **get_nine_patch_stretch**()

If `true`, Godot treats the bar's textures like in NinePatchRect. Use the `stretch_margin_*` properties like stretch_margin_bottom to set up the nine patch's 3×3 grid. When using a radial fill_mode, this setting will enable stretching.

Vector2 **radial_center_offset** = `Vector2(0, 0)` 🔗

- void **set_radial_center_offset**(value: Vector2)
- Vector2 **get_radial_center_offset**()

Offsets texture_progress if fill_mode is FILL_CLOCKWISE, FILL_COUNTER_CLOCKWISE, or FILL_CLOCKWISE_AND_COUNTER_CLOCKWISE.

---

float **radial_fill_degrees** = `360.0` 🔗

- void **set_fill_degrees**(value: float)
- float **get_fill_degrees**()

Upper limit for the fill of texture_progress if fill_mode is FILL_CLOCKWISE, FILL_COUNTER_CLOCKWISE, or FILL_CLOCKWISE_AND_COUNTER_CLOCKWISE. When the node's `value` is equal to its `max_value`, the texture fills up to this angle.

See Range.value, Range.max_value.

---

float **radial_initial_angle** = `0.0` 🔗

- void **set_radial_initial_angle**(value: float)

- float **get_radial_initial_angle**()

Starting angle for the fill of texture_progress if fill_mode is FILL_CLOCKWISE, FILL_COUNTER_CLOCKWISE, or FILL_CLOCKWISE_AND_COUNTER_CLOCKWISE. When the node's `value` is equal to its `min_value`, the texture doesn't show up at all. When the `value` increases, the texture fills and tends towards radial_fill_degrees.

---

int **stretch_margin_bottom** = `0` 🔗

- void **set_stretch_margin**(margin: Side, value: int)
- int **get_stretch_margin**(margin: Side) const

The height of the 9-patch's bottom row. A margin of 16 means the 9-slice's bottom corners and side will have a height of 16 pixels. You can set all 4 margin values individually to create panels with non-uniform borders. Only effective if nine_patch_stretch is `true`.

---

int **stretch_margin_left** = `0` 🔗

- void **set_stretch_margin**(margin: Side, value: int)
- int **get_stretch_margin**(margin: Side) const

The width of the 9-patch's left column. Only effective if nine_patch_stretch is `true`.

---

int **stretch_margin_right** = `0` 🔗

- void **set_stretch_margin**(margin: Side, value: int)
- int **get_stretch_margin**(margin: Side) const

The width of the 9-patch's right column. Only effective if nine_patch_stretch is `true`.

---

int **stretch_margin_top** = `0` 🔗

- void **set_stretch_margin**(margin: Side, value: int)
- int **get_stretch_margin**(margin: Side) const

The height of the 9-patch's top row. Only effective if nine_patch_stretch is `true`.

Texture2D **texture_over** 🔗

- void **set_over_texture**(value: Texture2D)
- Texture2D **get_over_texture**()

Texture2D that draws over the progress bar. Use it to add highlights or an upper-frame that hides part of texture_progress.

Texture2D **texture_progress** 🔗

- void **set_progress_texture**(value: Texture2D)
- Texture2D **get_progress_texture**()

Texture2D that clips based on the node's `value` and fill_mode. As `value` increased, the texture fills up. It shows entirely when `value` reaches `max_value`. It doesn't show at all if `value` is equal to `min_value`.

The `value` property comes from Range. See Range.value, Range.min_value, Range.max_value.

Vector2 **texture_progress_offset** = `Vector2(0, 0)` 🔗

- void **set_texture_progress_offset**(value: Vector2)
- Vector2 **get_texture_progress_offset**()

The offset of texture_progress. Useful for texture_over and texture_under with fancy borders, to avoid transparent margins in your progress texture.

Texture2D **texture_under** 🔗

- void **set_under_texture**(value: Texture2D)
- Texture2D **get_under_texture**()

Texture2D that draws under the progress bar. The bar's background.

Color **tint_over** = `Color(1, 1, 1, 1)` 🔗

- void **set_tint_over**(value: Color)
- Color **get_tint_over**()

Multiplies the color of the bar's texture_over texture. The effect is similar to CanvasItem.modulate, except it only affects this specific texture instead of the entire node.

Color **tint_progress** = `Color(1, 1, 1, 1)` 🔗

- void **set_tint_progress**(value: Color)
- Color **get_tint_progress**()

Multiplies the color of the bar's texture_progress texture.

Color **tint_under** = `Color(1, 1, 1, 1)` 🔗

- void **set_tint_under**(value: Color)
- Color **get_tint_under**()

Multiplies the color of the bar's texture_under texture.

Method Descriptions

int **get_stretch_margin**(margin: Side) const 🔗

Returns the stretch margin with the specified index. See stretch_margin_bottom and related properties.

---

void **set_stretch_margin**(margin: Side, value: int) 🔗

Sets the stretch margin with the specified index. See stretch_margin_bottom and related properties.

TextureRect

Inherits: Control **<** CanvasItem **<** Node **<** Object

A control that displays a texture.

Description

A control that displays a texture, for example an icon inside a GUI. The texture's placement can be controlled with the stretch_mode property. It can scale, tile, or stay centered inside its bounding rectangle.

Tutorials

- 3D Voxel Demo [https://godotengine.org/asset-library/asset/2755]

Properties

ExpandMode	expand_mode	`0`
bool	flip_h	`false`
bool	flip_v	`false`
MouseFilter	mouse_filter	`1` (overrides Control)

StretchMode	stretch_mode	`0`
Texture2D	texture	

Enumerations

enum **ExpandMode**: 🔗

ExpandMode **EXPAND_KEEP_SIZE** = `0`

The minimum size will be equal to texture size, i.e. **TextureRect** can't be smaller than the texture.

ExpandMode **EXPAND_IGNORE_SIZE** = `1`

The size of the texture won't be considered for minimum size calculation, so the **TextureRect** can be shrunk down past the texture size.

ExpandMode **EXPAND_FIT_WIDTH** = `2`

The height of the texture will be ignored. Minimum width will be equal to the current height. Useful for horizontal layouts, e.g. inside HBoxContainer.

ExpandMode **EXPAND_FIT_WIDTH_PROPORTIONAL** = `3`

Same as EXPAND_FIT_WIDTH, but keeps texture's aspect ratio.

ExpandMode **EXPAND_FIT_HEIGHT** = `4`

The width of the texture will be ignored. Minimum height will be equal to the current width. Useful for vertical layouts, e.g. inside

VBoxContainer.

ExpandMode **EXPAND_FIT_HEIGHT_PROPORTIONAL** = `5`

Same as EXPAND_FIT_HEIGHT, but keeps texture's aspect ratio.

---

enum **StretchMode**: 🔗

StretchMode **STRETCH_SCALE** = `0`

Scale to fit the node's bounding rectangle.

StretchMode **STRETCH_TILE** = `1`

Tile inside the node's bounding rectangle.

StretchMode **STRETCH_KEEP** = `2`

The texture keeps its original size and stays in the bounding rectangle's top-left corner.

StretchMode **STRETCH_KEEP_CENTERED** = `3`

The texture keeps its original size and stays centered in the node's bounding rectangle.

StretchMode **STRETCH_KEEP_ASPECT** = `4`

Scale the texture to fit the node's bounding rectangle, but maintain the texture's aspect ratio.

StretchMode **STRETCH_KEEP_ASPECT_CENTERED** = `5`

Scale the texture to fit the node's bounding rectangle, center it and maintain its aspect ratio.

StretchMode **STRETCH_KEEP_ASPECT_COVERED** = `6`

Scale the texture so that the shorter side fits the bounding rectangle. The other side clips to the node's limits.

Property Descriptions

ExpandMode **expand_mode** = `0` 🔗

- void **set_expand_mode**(value: ExpandMode)
- ExpandMode **get_expand_mode**()

Experimental: Using EXPAND_FIT_WIDTH, EXPAND_FIT_WIDTH_PROPORTIONAL, EXPAND_FIT_HEIGHT, or EXPAND_FIT_HEIGHT_PROPORTIONAL may result in unstable behavior in some Container controls. This behavior may be re-evaluated and changed in the future.

Defines how minimum size is determined based on the texture's size. See ExpandMode for options.

---

bool **flip_h** = `false` 🔗

- void **set_flip_h**(value: bool)
- bool **is_flipped_h**()

If `true`, texture is flipped horizontally.

---

bool **flip_v** = `false` 🔗

- void **set_flip_v**(value: bool)
- bool **is_flipped_v**()

If `true`, texture is flipped vertically.

StretchMode **stretch_mode** = `0` 🔗

- void **set_stretch_mode**(value: StretchMode)
- StretchMode **get_stretch_mode**()

Controls the texture's behavior when resizing the node's bounding rectangle. See StretchMode.

Texture2D **texture** 🔗

- void **set_texture**(value: Texture2D)
- Texture2D **get_texture**()

The node's Texture2D resource.

TileMap

Deprecated: Use multiple TileMapLayer nodes instead. To convert a TileMap to a set of TileMapLayer nodes, open the TileMap bottom panel with the node selected, click the toolbox icon in the top-right corner and choose 'Extract TileMap layers as individual TileMapLayer nodes'.

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Node for 2D tile-based maps.

Description

Node for 2D tile-based maps. Tilemaps use a TileSet which contain a list of tiles which are used to create grid-based maps. A TileMap may have several layers, layouting tiles on top of each other.

For performance reasons, all TileMap updates are batched at the end of a frame. Notably, this means that scene tiles from a TileSetScenesCollectionSource may be initialized after their parent. This is only queued when inside the scene tree.

To force an update earlier on, call update_internals.

Tutorials

- Using Tilemaps
- 2D Platformer Demo [https://godotengine.org/asset-library/asset/2727]
- 2D Isometric Demo [https://godotengine.org/asset-library/asset/2718]
- 2D Hexagonal Demo [https://godotengine.org/asset-library/asset/2717]

- 2D Grid-based Navigation with AStarGrid2D Demo [https://godotengine.org/asset-library/asset/2723]
- 2D Role Playing Game (RPG) Demo [https://godotengine.org/asset-library/asset/2729]
- 2D Kinematic Character Demo [https://godotengine.org/asset-library/asset/2719]

Properties

bool	collision_animatable	`false`
VisibilityMode	collision_visibility_mode	`0`
VisibilityMode	navigation_visibility_mode	`0`
int	rendering_quadrant_size	`16`
TileSet	tile_set	

Methods

void	_tile_data_runtime_update(layer: int, coords: Vector2i, tile_data: TileData) virtual
bool	_use_tile_data_runtime_update(layer: int, coords: Vector2i) virtual

void	add_layer(to_position: int)
void	clear()
void	clear_layer(layer: int)
void	erase_cell(layer: int, coords: Vector2i)
void	fix_invalid_tiles()
void	force_update(layer: int = -1)
int	get_cell_alternative_tile(layer: int, coords: Vector2i, use_proxies: bool = false) const
Vector2i	get_cell_atlas_coords(layer: int, coords: Vector2i, use_proxies: bool = false) const
int	get_cell_source_id(layer: int, coords: Vector2i, use_proxies: bool = false) const
TileData	get_cell_tile_data(layer: int, coords: Vector2i, use_proxies: bool = false) const
Vector2i	get_coords_for_body_rid(body: RID)

int	get_layer_for_body_rid(body: RID)
Color	get_layer_modulate(layer: int) const
String	get_layer_name(layer: int) const
RID	get_layer_navigation_map(layer: int) const
int	get_layer_y_sort_origin(layer: int) const
int	get_layer_z_index(layer: int) const
int	get_layers_count() const
RID	get_navigation_map(layer: int) const
Vector2i	get_neighbor_cell(coords: Vector2i, neighbor: CellNeighbor) const
TileMapPattern	get_pattern(layer: int, coords_array: Array[Vector2i])
Array[Vector2i]	get_surrounding_cells(coords: Vector2i)
Array[Vector2i]	get_used_cells(layer: int) const

Array[Vector2i]	get_used_cells_by_id(layer: int, source_id: int = -1, atlas_coords: Vector2i = Vector2i(-1, -1), alternative_tile: int = -1) const
Rect2i	get_used_rect() const
bool	is_layer_enabled(layer: int) const
bool	is_layer_navigation_enabled(layer: int) const
bool	is_layer_y_sort_enabled(layer: int) const
Vector2i	local_to_map(local_position: Vector2) const
Vector2i	map_pattern(position_in_tilemap: Vector2i, coords_in_pattern: Vector2i, pattern: TileMapPattern)
Vector2	map_to_local(map_position: Vector2i) const
void	move_layer(layer: int, to_position: int)
void	notify_runtime_tile_data_update(layer: int = -1)
void	remove_layer(layer: int)

void	set_cell(layer: int, coords: Vector2i, source_id: int = -1, atlas_coords: Vector2i = Vector2i(-1, -1), alternative_tile: int = 0)
void	set_cells_terrain_connect(layer: int, cells: Array[Vector2i], terrain_set: int, terrain: int, ignore_empty_terrains: bool = true)
void	set_cells_terrain_path(layer: int, path: Array[Vector2i], terrain_set: int, terrain: int, ignore_empty_terrains: bool = true)
void	set_layer_enabled(layer: int, enabled: bool)
void	set_layer_modulate(layer: int, modulate: Color)
void	set_layer_name(layer: int, name: String)
void	set_layer_navigation_enabled(layer: int, enabled: bool)
void	set_layer_navigation_map(layer: int, map: RID)
void	set_layer_y_sort_enabled(layer: int, y_sort_enabled: bool)

void	set_layer_y_sort_origin(layer: int, y_sort_origin: int)
void	set_layer_z_index(layer: int, z_index: int)
void	set_navigation_map(layer: int, map: RID)
void	set_pattern(layer: int, position: Vector2i, pattern: TileMapPattern)
void	update_internals()

Signals

changed() 🔗

Emitted when the TileSet of this TileMap changes.

Enumerations

enum **VisibilityMode**: 🔗

VisibilityMode **VISIBILITY_MODE_DEFAULT** = `0`

Use the debug settings to determine visibility.

VisibilityMode **VISIBILITY_MODE_FORCE_HIDE** = `2`

Always hide.

VisibilityMode **VISIBILITY_MODE_FORCE_SHOW** = `1`

Always show.

Property Descriptions

bool **collision_animatable** = `false`

- void **set_collision_animatable**(value: bool)
- bool **is_collision_animatable**()

If enabled, the TileMap will see its collisions synced to the physics tick and change its collision type from static to kinematic. This is required to create TileMap-based moving platform.

Note: Enabling collision_animatable may have a small performance impact, only do it if the TileMap is moving and has colliding tiles.

VisibilityMode **collision_visibility_mode** = `0`

- void **set_collision_visibility_mode**(value: VisibilityMode)
- VisibilityMode **get_collision_visibility_mode**()

Show or hide the TileMap's collision shapes. If set to VISIBILITY_MODE_DEFAULT, this depends on the show collision debug settings.

VisibilityMode **navigation_visibility_mode** = `0`

- void **set_navigation_visibility_mode**(value: VisibilityMode)
- VisibilityMode **get_navigation_visibility_mode**()

Show or hide the TileMap's navigation meshes. If set to VISIBILITY_MODE_DEFAULT, this depends on the show navigation debug settings.

---

int **rendering_quadrant_size** = `16` 🔗

- void **set_rendering_quadrant_size**(value: int)
- int **get_rendering_quadrant_size**()

The TileMap's quadrant size. A quadrant is a group of tiles to be drawn together on a single canvas item, for optimization purposes. rendering_quadrant_size defines the length of a square's side, in the map's coordinate system, that forms the quadrant. Thus, the default quadrant size groups together `16 * 16 = 256` tiles.

The quadrant size does not apply on Y-sorted layers, as tiles are grouped by Y position instead in that case.

Note: As quadrants are created according to the map's coordinate system, the quadrant's "square shape" might not look like square in the TileMap's local coordinate system.

TileSet **tile_set** 🔗

- void **set_tileset**(value: TileSet)
- TileSet **get_tileset**()

The TileSet used by this **TileMap**. The textures, collisions, and additional behavior of all available tiles are stored here.

Method Descriptions

void **_tile_data_runtime_update**(layer: int, coords: Vector2i, tile_data: TileData) virtual 🔗

Called with a TileData object about to be used internally by the TileMap, allowing its modification at runtime.

This method is only called if _use_tile_data_runtime_update is implemented and returns `true` for the given tile `coords` and `layer`.

Warning: The `tile_data` object's sub-resources are the same as the one in the TileSet. Modifying them might impact the whole TileSet. Instead, make sure to duplicate those resources.

Note: If the properties of `tile_data` object should change over time, use notify_runtime_tile_data_update to notify the TileMap it needs an update.

---

bool **_use_tile_data_runtime_update**(layer: int, coords: Vector2i) virtual 🔗

Should return `true` if the tile at coordinates `coords` on layer `layer` requires a runtime update.

Warning: Make sure this function only return `true` when needed. Any tile processed at runtime without a need for it will imply a significant performance penalty.

Note: If the result of this function should changed, use notify_runtime_tile_data_update to notify the TileMap it needs an update.

---

void **add_layer**(to_position: int) 🔗

Adds a layer at the given position `to_position` in the array. If `to_position` is negative, the position is counted from the end, with

`-1` adding the layer at the end of the array.

---

void **clear**()

Clears all cells.

---

void **clear_layer**(layer: int)

Clears all cells on the given layer.

If `layer` is negative, the layers are accessed from the last one.

---

void **erase_cell**(layer: int, coords: Vector2i)

Erases the cell on layer `layer` at coordinates `coords`.

If `layer` is negative, the layers are accessed from the last one.

---

void **fix_invalid_tiles**()

Clears cells that do not exist in the tileset.

---

void **force_update**(layer: int = -1)

Deprecated: Use notify_runtime_tile_data_update and/or update_internals instead.

Forces the TileMap and the layer `layer` to update.

---

int **get_cell_alternative_tile**(layer: int, coords: Vector2i, use_proxies: bool = false) const

Returns the tile alternative ID of the cell on layer `layer` at `coords`.

If `use_proxies` is `false`, ignores the TileSet's tile proxies, returning the raw alternative identifier. See TileSet.map_tile_proxy.

If `layer` is negative, the layers are accessed from the last one.

Vector2i **get_cell_atlas_coords**(layer: int, coords: Vector2i, use_proxies: bool = false) const 🔗

Returns the tile atlas coordinates ID of the cell on layer `layer` at coordinates `coords`. Returns `Vector2i(-1, -1)` if the cell does not exist.

If `use_proxies` is `false`, ignores the TileSet's tile proxies, returning the raw atlas coordinate identifier. See TileSet.map_tile_proxy.

If `layer` is negative, the layers are accessed from the last one.

---

int **get_cell_source_id**(layer: int, coords: Vector2i, use_proxies: bool = false) const 🔗

Returns the tile source ID of the cell on layer `layer` at coordinates `coords`. Returns `-1` if the cell does not exist.

If `use_proxies` is `false`, ignores the TileSet's tile proxies, returning the raw source identifier. See TileSet.map_tile_proxy.

If `layer` is negative, the layers are accessed from the last one.

TileData **get_cell_tile_data**(layer: int, coords: Vector2i, use_proxies: bool = false) const 🔗

Returns the TileData object associated with the given cell, or `null` if the cell does not exist or is not a TileSetAtlasSource.

If `layer` is negative, the layers are accessed from the last one.

```
func get_clicked_tile_power():
    var clicked_cell = tile_map.local_to_map(tile_map.get_local_mouse_position())
    var data = tile_map.get_cell_tile_data(0, clicked_cell)
    if data:
        return data.get_custom_data("power")
    else:
        return 0
```

If `use_proxies` is `false`, ignores the TileSet's tile proxies. See TileSet.map_tile_proxy.

Vector2i **get_coords_for_body_rid**(body: RID) 🔗

Returns the coordinates of the tile for given physics body RID. Such RID can be retrieved from KinematicCollision2D.get_collider_rid, when colliding with a tile.

---

int **get_layer_for_body_rid**(body: RID) 🔗

Returns the tilemap layer of the tile for given physics body RID. Such RID can be retrieved from KinematicCollision2D.get_collider_rid, when colliding with a tile.

Color **get_layer_modulate**(layer: int) const 🔗

Returns a TileMap layer's modulate.

If `layer` is negative, the layers are accessed from the last one.

String **get_layer_name**(layer: int) const

Returns a TileMap layer's name.

If `layer` is negative, the layers are accessed from the last one.

RID **get_layer_navigation_map**(layer: int) const

Returns the RID of the NavigationServer2D navigation map assigned to the specified TileMap layer `layer`.

By default the TileMap uses the default World2D navigation map for the first TileMap layer. For each additional TileMap layer a new navigation map is created for the additional layer.

In order to make NavigationAgent2D switch between TileMap layer navigation maps use NavigationAgent2D.set_navigation_map with the navigation map received from get_layer_navigation_map.

If `layer` is negative, the layers are accessed from the last one.

---

int **get_layer_y_sort_origin**(layer: int) const

Returns a TileMap layer's Y sort origin.

If `layer` is negative, the layers are accessed from the last one.

---

int **get_layer_z_index**(layer: int) const

Returns a TileMap layer's Z-index value.

If `layer` is negative, the layers are accessed from the last one.

---

int **get_layers_count**() const

Returns the number of layers in the TileMap.

RID **get_navigation_map**(layer: int) const

Deprecated: Use get_layer_navigation_map instead.

Returns the RID of the NavigationServer2D navigation map assigned to the specified TileMap layer `layer`.

Vector2i **get_neighbor_cell**(coords: Vector2i, neighbor: CellNeighbor) const

Returns the neighboring cell to the one at coordinates `coords`, identified by the `neighbor` direction. This method takes into account the different layouts a TileMap can take.

TileMapPattern **get_pattern**(layer: int, coords_array: Array[Vector2i])

Creates a new TileMapPattern from the given layer and set of cells.

If `layer` is negative, the layers are accessed from the last one.

Array[Vector2i] **get_surrounding_cells**(coords: Vector2i)

Returns the list of all neighbourings cells to the one at `coords`.

Array[Vector2i] **get_used_cells**(layer: int) const

Returns a Vector2i array with the positions of all cells containing a tile in the given layer. A cell is considered empty if its source identifier equals -1, its atlas coordinates identifiers is `Vector2(-1, -1)` and its alternative identifier is -1.

If `layer` is negative, the layers are accessed from the last one.

Array[Vector2i] **get_used_cells_by_id**(layer: int, source_id: int = -1, atlas_coords: Vector2i = Vector2i(-1, -1), alternative_tile: int = -1) const 🔗

Returns a Vector2i array with the positions of all cells containing a tile in the given layer. Tiles may be filtered according to their source (`source_id`), their atlas coordinates (`atlas_coords`) or alternative id (`alternative_tile`).

If a parameter has its value set to the default one, this parameter is not used to filter a cell. Thus, if all parameters have their respective default value, this method returns the same result as get_used_cells.

A cell is considered empty if its source identifier equals -1, its atlas coordinates identifiers is `Vector2(-1, -1)` and its alternative identifier is -1.

If `layer` is negative, the layers are accessed from the last one.

Rect2i **get_used_rect**() const 🔗

Returns a rectangle enclosing the used (non-empty) tiles of the map, including all layers.

---

bool **is_layer_enabled**(layer: int) const 🔗

Returns if a layer is enabled.

If `layer` is negative, the layers are accessed from the last one.

bool **is_layer_navigation_enabled**(layer: int) const

Returns if a layer's built-in navigation regions generation is enabled.

bool **is_layer_y_sort_enabled**(layer: int) const

Returns if a layer Y-sorts its tiles.

If `layer` is negative, the layers are accessed from the last one.

Vector2i **local_to_map**(local_position: Vector2) const

Returns the map coordinates of the cell containing the given `local_position`. If `local_position` is in global coordinates, consider using Node2D.to_local before passing it to this method. See also map_to_local.

Vector2i **map_pattern**(position_in_tilemap: Vector2i, coords_in_pattern: Vector2i, pattern: TileMapPattern)

Returns for the given coordinate `coords_in_pattern` in a TileMapPattern the corresponding cell coordinates if the pattern was pasted at the `position_in_tilemap` coordinates (see set_pattern). This mapping is required as in half-offset tile shapes, the mapping might not work by calculating `position_in_tile_map + coords_in_pattern`.

Vector2 **map_to_local**(map_position: Vector2i) const

Returns the centered position of a cell in the TileMap's local coordinate space. To convert the returned value into global coordinates, use Node2D.to_global. See also local_to_map.

Note: This may not correspond to the visual position of the tile, i.e. it ignores the TileData.texture_origin property of individual tiles.

void **move_layer**(layer: int, to_position: int) 🔗

Moves the layer at index `layer` to the given position `to_position` in the array.

void **notify_runtime_tile_data_update**(layer: int = -1) 🔗

Notifies the TileMap node that calls to _use_tile_data_runtime_update or _tile_data_runtime_update will lead to different results. This will thus trigger a TileMap update.

If `layer` is provided, only notifies changes for the given layer. Providing the `layer` argument (when applicable) is usually preferred for performance reasons.

Warning: Updating the TileMap is computationally expensive and may impact performance. Try to limit the number of calls to this function to avoid unnecessary update.

Note: This does not trigger a direct update of the TileMap, the update will be done at the end of the frame as usual (unless you call update_internals).

void **remove_layer**(layer: int) 🔗

Removes the layer at index `layer`.

void **set_cell**(layer: int, coords: Vector2i, source_id: int = -1, atlas_coords: Vector2i = Vector2i(-1, -1), alternative_tile: int = 0) 🔗

Sets the tile identifiers for the cell on layer `layer` at coordinates `coords`. Each tile of the TileSet is identified using three parts:

- The source identifier `source_id` identifies a TileSetSource identifier. See TileSet.set_source_id,
- The atlas coordinates identifier `atlas_coords` identifies a tile coordinates in the atlas (if the source is a TileSetAtlasSource). For TileSetScenesCollectionSource it should always be `Vector2i(0, 0)`),
- The alternative tile identifier `alternative_tile` identifies a tile alternative in the atlas (if the source is a TileSetAtlasSource), and the scene for a TileSetScenesCollectionSource.

If `source_id` is set to `-1`, `atlas_coords` to `Vector2i(-1, -1)` or `alternative_tile` to `-1`, the cell will be erased. An erased cell gets **all** its identifiers automatically set to their respective invalid values, namely `-1`, `Vector2i(-1, -1)` and `-1`.

If `layer` is negative, the layers are accessed from the last one.

---

void **set_cells_terrain_connect**(layer: int, cells: Array[Vector2i], terrain_set: int, terrain: int, ignore_empty_terrains: bool = true) 🔗

Update all the cells in the `cells` coordinates array so that they use the given `terrain` for the given `terrain_set`. If an updated cell has the same terrain as one of its neighboring cells, this function tries to join the two. This function might update neighboring tiles if needed to create correct terrain transitions.

If `ignore_empty_terrains` is true, empty terrains will be ignored when trying to find the best fitting tile for the given terrain constraints.

If `layer` is negative, the layers are accessed from the last one.

Note: To work correctly, this method requires the TileMap's TileSet to have terrains set up with all required terrain combinations. Otherwise, it may produce unexpected results.

void **set_cells_terrain_path**(layer: int, path: Array[Vector2i], terrain_set: int, terrain: int, ignore_empty_terrains: bool = true)

Update all the cells in the `path` coordinates array so that they use the given `terrain` for the given `terrain_set`. The function will also connect two successive cell in the path with the same terrain. This function might update neighboring tiles if needed to create correct terrain transitions.

If `ignore_empty_terrains` is true, empty terrains will be ignored when trying to find the best fitting tile for the given terrain constraints.

If `layer` is negative, the layers are accessed from the last one.

Note: To work correctly, this method requires the TileMap's TileSet to have terrains set up with all required terrain combinations. Otherwise, it may produce unexpected results.

void **set_layer_enabled**(layer: int, enabled: bool)

Enables or disables the layer `layer`. A disabled layer is not processed at all (no rendering, no physics, etc.).

If `layer` is negative, the layers are accessed from the last one.

void **set_layer_modulate**(layer: int, modulate: Color) 🔗

Sets a layer's color. It will be multiplied by tile's color and TileMap's modulate.

If `layer` is negative, the layers are accessed from the last one.

---

void **set_layer_name**(layer: int, name: String) 🔗

Sets a layer's name. This is mostly useful in the editor.

If `layer` is negative, the layers are accessed from the last one.

---

void **set_layer_navigation_enabled**(layer: int, enabled: bool) 🔗

Enables or disables a layer's built-in navigation regions generation. Disable this if you need to bake navigation regions from a TileMap using a NavigationRegion2D node.

---

void **set_layer_navigation_map**(layer: int, map: RID) 🔗

Assigns `map` as a NavigationServer2D navigation map for the specified TileMap layer `layer`.

By default the TileMap uses the default World2D navigation map for the first TileMap layer. For each additional TileMap layer a new navigation map is created for the additional layer.

In order to make NavigationAgent2D switch between TileMap layer navigation maps use NavigationAgent2D.set_navigation_map with the navigation map received from get_layer_navigation_map.

If `layer` is negative, the layers are accessed from the last one.

---

void **set_layer_y_sort_enabled**(layer: int, y_sort_enabled: bool) 🔗

Enables or disables a layer's Y-sorting. If a layer is Y-sorted, the layer will behave as a CanvasItem node where each of its tile gets Y-sorted.

Y-sorted layers should usually be on different Z-index values than not Y-sorted layers, otherwise, each of those layer will be Y-sorted as whole with the Y-sorted one. This is usually an undesired behavior.

If `layer` is negative, the layers are accessed from the last one.

---

void **set_layer_y_sort_origin**(layer: int, y_sort_origin: int) 🔗

Sets a layer's Y-sort origin value. This Y-sort origin value is added to each tile's Y-sort origin value.

This allows, for example, to fake a different height level on each layer. This can be useful for top-down view games.

If `layer` is negative, the layers are accessed from the last one.

---

void **set_layer_z_index**(layer: int, z_index: int) 🔗

Sets a layers Z-index value. This Z-index is added to each tile's Z-index value.

If `layer` is negative, the layers are accessed from the last one.

---

void **set_navigation_map**(layer: int, map: RID) 🔗

Deprecated: Use set_layer_navigation_map instead.

Assigns `map` as a NavigationServer2D navigation map for the specified TileMap layer `layer`.

---

void **set_pattern**(layer: int, position: Vector2i, pattern: TileMapPattern) 🔗

Paste the given TileMapPattern at the given `position` and `layer` in the tile map.

If `layer` is negative, the layers are accessed from the last one.

---

void **update_internals**() 🔗

Triggers a direct update of the TileMap. Usually, calling this function is not needed, as TileMap node updates automatically when one of its properties or cells is modified.

However, for performance reasons, those updates are batched and delayed to the end of the frame. Calling this function will force the TileMap to update right away instead.

Warning: Updating the TileMap is computationally expensive and may impact performance. Try to limit the number of updates and how many tiles they impact.

TileMapLayer

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Node for 2D tile-based maps.

Description

Node for 2D tile-based maps. A **TileMapLayer** uses a TileSet which contain a list of tiles which are used to create grid-based maps. Unlike the TileMap node, which is deprecated, **TileMapLayer** has only one layer of tiles. You can use several **TileMapLayer** to achieve the same result as a TileMap node.

For performance reasons, all TileMap updates are batched at the end of a frame. Notably, this means that scene tiles from a TileSetScenesCollectionSource may be initialized after their parent. This is only queued when inside the scene tree.

To force an update earlier on, call update_internals.

Properties

bool	collision_enabled	`true`
DebugVisibilityMode	collision_visibility_mode	`0`
bool	enabled	`true`

bool	navigation_enabled	`true`
DebugVisibilityMode	navigation_visibility_mode	`0`
int	rendering_quadrant_size	`16`
PackedByteArray	tile_map_data	`PackedByteArray()`
TileSet	tile_set	
bool	use_kinematic_bodies	`false`
bool	x_draw_order_reversed	`false`
int	y_sort_origin	`0`

Methods

void	_tile_data_runtime_update(coords: Vector2i, tile_data: TileData) virtual
bool	_use_tile_data_runtime_update(coords: Vector2i) virtual

void	clear()
void	erase_cell(coords: Vector2i)
void	fix_invalid_tiles()
int	get_cell_alternative_tile(coords: Vector2i) const
Vector2i	get_cell_atlas_coords(coords: Vector2i) const
int	get_cell_source_id(coords: Vector2i) const
TileData	get_cell_tile_data(coords: Vector2i) const
Vector2i	get_coords_for_body_rid(body: RID) const
RID	get_navigation_map() const
Vector2i	get_neighbor_cell(coords: Vector2i, neighbor: CellNeighbor) const
TileMapPattern	get_pattern(coords_array: Array[Vector2i])
Array[Vector2i]	get_surrounding_cells(coords: Vector2i)

Array[Vector2i]	get_used_cells() const
Array[Vector2i]	get_used_cells_by_id(source_id: int = -1, atlas_coords: Vector2i = Vector2i(-1, -1), alternative_tile: int = -1) const
Rect2i	get_used_rect() const
bool	has_body_rid(body: RID) const
Vector2i	local_to_map(local_position: Vector2) const
Vector2i	map_pattern(position_in_tilemap: Vector2i, coords_in_pattern: Vector2i, pattern: TileMapPattern)
Vector2	map_to_local(map_position: Vector2i) const
void	notify_runtime_tile_data_update()
void	set_cell(coords: Vector2i, source_id: int = -1, atlas_coords: Vector2i = Vector2i(-1, -1), alternative_tile: int = 0)
void	set_cells_terrain_connect(cells: Array[Vector2i], terrain_set: int, terrain: int, ignore_empty_terrains: bool = true)

void	set_cells_terrain_path(path: Array[Vector2i], terrain_set: int, terrain: int, ignore_empty_terrains: bool = true)
void	set_navigation_map(map: RID)
void	set_pattern(position: Vector2i, pattern: TileMapPattern)
void	update_internals()

Signals

changed() 🔗

Emitted when this **TileMapLayer**'s properties changes. This includes modified cells, properties, or changes made to its assigned TileSet.

Note: This signal may be emitted very often when batch-modifying a **TileMapLayer**. Avoid executing complex processing in a connected function, and consider delaying it to the end of the frame instead (i.e. calling Object.call_deferred).

Enumerations

enum **DebugVisibilityMode**: 🔗

DebugVisibilityMode **DEBUG_VISIBILITY_MODE_DEFAULT** = `0`

Hide the collisions or navigation debug shapes in the editor, and use the debug settings to determine their visibility in game (i.e. SceneTree.debug_collisions_hint or SceneTree.debug_navigation_hint).

DebugVisibilityMode **DEBUG_VISIBILITY_MODE_FORCE_HIDE** = `2`

Always hide the collisions or navigation debug shapes.

DebugVisibilityMode **DEBUG_VISIBILITY_MODE_FORCE_SHOW** = `1`

Always show the collisions or navigation debug shapes.

Property Descriptions

bool **collision_enabled** = `true`

- void **set_collision_enabled**(value: bool)
- bool **is_collision_enabled**()

Enable or disable collisions.

DebugVisibilityMode **collision_visibility_mode** = `0`

- void **set_collision_visibility_mode**(value: DebugVisibilityMode)
- DebugVisibilityMode **get_collision_visibility_mode**()

Show or hide the **TileMapLayer**'s collision shapes. If set to DEBUG_VISIBILITY_MODE_DEFAULT, this depends on the show collision debug settings.

---

bool **enabled** = `true` 🔗

- void **set_enabled**(value: bool)
- bool **is_enabled**()

If `false`, disables this **TileMapLayer** completely (rendering, collision, navigation, scene tiles, etc.)

---

bool **navigation_enabled** = `true` 🔗

- void **set_navigation_enabled**(value: bool)
- bool **is_navigation_enabled**()

If `true`, navigation regions are enabled.

DebugVisibilityMode **navigation_visibility_mode** = `0` 🔗

- void **set_navigation_visibility_mode**(value: DebugVisibilityMode)
- DebugVisibilityMode **get_navigation_visibility_mode**()

Show or hide the **TileMapLayer**'s navigation meshes. If set to DEBUG_VISIBILITY_MODE_DEFAULT, this depends on the show navigation debug settings.

---

int **rendering_quadrant_size** = `16` 🔗

- void **set_rendering_quadrant_size**(value: int)
- int **get_rendering_quadrant_size**()

The **TileMapLayer**'s quadrant size. A quadrant is a group of tiles to be drawn together on a single canvas item, for optimization purposes. rendering_quadrant_size defines the length of a square's side, in the map's coordinate system, that forms the quadrant. Thus, the default quadrant size groups together `16 * 16 = 256` tiles.

The quadrant size does not apply on a Y-sorted **TileMapLayer**, as tiles are grouped by Y position instead in that case.

Note: As quadrants are created according to the map's coordinate system, the quadrant's "square shape" might not look like square in the **TileMapLayer**'s local coordinate system.

PackedByteArray **tile_map_data** = `PackedByteArray()` 🔗

- void **set_tile_map_data_from_array**(value: PackedByteArray)
- PackedByteArray **get_tile_map_data_as_array**()

The raw tile map data as a byte array.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedByteArray for more details.

TileSet **tile_set** 🔗

- void **set_tile_set**(value: TileSet)
- TileSet **get_tile_set**()

The TileSet used by this layer. The textures, collisions, and additional behavior of all available tiles are stored here.

bool **use_kinematic_bodies** = `false` 🔗

- void **set_use_kinematic_bodies**(value: bool)
- bool **is_using_kinematic_bodies**()

If `true`, this **TileMapLayer** collision shapes will be instantiated as kinematic bodies. This can be needed for moving **TileMapLayer** nodes (i.e. moving platforms).

---

bool **x_draw_order_reversed** = `false` 🔗

- void **set_x_draw_order_reversed**(value: bool)
- bool **is_x_draw_order_reversed**()

If CanvasItem.y_sort_enabled is enabled, setting this to `true` will reverse the order the tiles are drawn on the X-axis.

---

int **y_sort_origin** = `0` 🔗

- void **set_y_sort_origin**(value: int)
- int **get_y_sort_origin**()

This Y-sort origin value is added to each tile's Y-sort origin value. This allows, for example, to fake a different height level. This can be useful for top-down view games.

Method Descriptions

void **_tile_data_runtime_update**(coords: Vector2i, tile_data: TileData) virtual 🔗

Called with a TileData object about to be used internally by the **TileMapLayer**, allowing its modification at runtime.

This method is only called if _use_tile_data_runtime_update is implemented and returns `true` for the given tile `coords`.

Warning: The `tile_data` object's sub-resources are the same as the one in the TileSet. Modifying them might impact the whole TileSet. Instead, make sure to duplicate those resources.

Note: If the properties of `tile_data` object should change over time, use notify_runtime_tile_data_update to notify the **TileMapLayer** it needs an update.

---

bool **_use_tile_data_runtime_update**(coords: Vector2i) virtual

Should return `true` if the tile at coordinates `coords` requires a runtime update.

Warning: Make sure this function only returns `true` when needed. Any tile processed at runtime without a need for it will imply a significant performance penalty.

Note: If the result of this function should change, use notify_runtime_tile_data_update to notify the **TileMapLayer** it needs an update.

---

void **clear**()

Clears all cells.

---

void **erase_cell**(coords: Vector2i)

Erases the cell at coordinates `coords`.

---

void **fix_invalid_tiles**()

Clears cells containing tiles that do not exist in the tile_set.

int **get_cell_alternative_tile**(coords: Vector2i) const

Returns the tile alternative ID of the cell at coordinates `coords`.

Vector2i **get_cell_atlas_coords**(coords: Vector2i) const

Returns the tile atlas coordinates ID of the cell at coordinates `coords`. Returns `Vector2i(-1, -1)` if the cell does not exist.

int **get_cell_source_id**(coords: Vector2i) const

Returns the tile source ID of the cell at coordinates `coords`. Returns `-1` if the cell does not exist.

TileData **get_cell_tile_data**(coords: Vector2i) const

Returns the TileData object associated with the given cell, or `null` if the cell does not exist or is not a TileSetAtlasSource.

```
func get_clicked_tile_power():
    var clicked_cell = tile_map_layer.local_to_map(tile_map_layer.get_local_mouse_position())
    var data = tile_map_layer.get_cell_tile_data(clicked_cell)
    if data:
        return data.get_custom_data("power")
    else:
        return 0
```

Vector2i **get_coords_for_body_rid**(body: RID) const

Returns the coordinates of the tile for given physics body RID. Such an RID can be retrieved from KinematicCollision2D.get_collider_rid, when colliding with a tile.

RID **get_navigation_map**() const

Returns the RID of the NavigationServer2D navigation used by this **TileMapLayer**.

By default this returns the default World2D navigation map, unless a custom map was provided using set_navigation_map.

Vector2i **get_neighbor_cell**(coords: Vector2i, neighbor: CellNeighbor) const

Returns the neighboring cell to the one at coordinates `coords`, identified by the `neighbor` direction. This method takes into account the different layouts a TileMap can take.

TileMapPattern **get_pattern**(coords_array: Array[Vector2i])

Creates and returns a new TileMapPattern from the given array of cells. See also set_pattern.

Array[Vector2i] **get_surrounding_cells**(coords: Vector2i)

Returns the list of all neighboring cells to the one at `coords`.

Array[Vector2i] **get_used_cells**() const

Returns a Vector2i array with the positions of all cells containing a tile. A cell is considered empty if its source identifier equals `-1`, its atlas coordinate identifier is `Vector2(-1, -1)` and its alternative identifier is `-1`.

Array[Vector2i] **get_used_cells_by_id**(source_id: int = -1, atlas_coords: Vector2i = Vector2i(-1, -1), alternative_tile: int = -1) const

Returns a Vector2i array with the positions of all cells containing a tile. Tiles may be filtered according to their source (`source_id`), their atlas coordinates (`atlas_coords`), or alternative id (`alternative_tile`).

If a parameter has its value set to the default one, this parameter is not used to filter a cell. Thus, if all parameters have their respective default values, this method returns the same result as get_used_cells.

A cell is considered empty if its source identifier equals `-1`, its atlas coordinate identifier is `Vector2(-1, -1)` and its alternative identifier is `-1`.

Rect2i **get_used_rect**() const

Returns a rectangle enclosing the used (non-empty) tiles of the map.

bool **has_body_rid**(body: RID) const

Returns whether the provided `body` RID belongs to one of this **TileMapLayer**'s cells.